Quantum algorithms for matrix inversion

Kaczúr Flórián

Thesis

Mathematics Msc

2023

Supervisor:
Friedl Katalin BME-VIK

Inner supervisor:
Grolmusz Vince ELTE-TTK

Eötvös Loránd University

Faculty of Sciences

NYILATKOZAT

Név: Kaczúr Flórián

ELTE Természettudományi Kar, szak: Matematikus Msc

NEPTUN azonosító: FL80LW

Diplomamunka címe:
Quantum algorithms for matrix inversion

A **diplomamunka** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2023. 06. 05.

[illegible]

a hallgató aláírása

Acknowledgement

First and foremost, I would like to thank my supervisor for her regular guidance. The thesis in this form couldn't have been made without her. I am deeply grateful to my family for their constant encouragement and support. Last but not least, I would like to express my gratitude to my friends who not only supported me in my studies but also shared numerous memorable experiences with me.

Contents

Introduction

Quantum computing is a rapidly evolving field at the forefront of scientific research and technological advancements. Unlike classical computing, which relies on bits represented as 0s and 1s, quantum computing harnesses the unique properties of quantum systems, such as superposition and entanglement, to perform computations with quantum bits or qubits. For this reason, quantum computing differs from classical computing due to the underlying physics involved, which necessitates the development of new mathematical frameworks. The mathematical foundations of this area were laid in the late 20th century, during which significant discoveries were made, such as Shor's algorithm and Grover's algorithm. These notable breakthroughs demonstrated the potential of quantum computers to outperform classical counterparts in specific computational tasks. Interestingly, it is proven that quantum computers can solve the same problems as classical computers, but with the advantage of potentially providing faster solutions for certain problems.

The central goal of quantum computing research is twofold. First, it aims to engineer and scale up robust and stable quantum systems with a high number of qubits. Building a reliable quantum computer is a formidable engineering challenge due to the delicate nature of quantum systems and the need for precise control and mitigation of errors. Second, researchers strive to design and develop quantum algorithms that exploit the unique capabilities of quantum computers, leading to new approaches and insights across various scientific and technological domains.

At the present time, the best universal quantum computers don't exceed the limits of classical computation, although a few special-purpose quantum machines can work much better for some specific problems. Many times, when applying classical methods to massive-scale dataset, we must sacrifice accuracy in exchange for speed. There is high hope that with the help of quantum computing we can overcome this trade-off. With a fast and accurate method in hand, one can make significantly better predictions.

The major motivation for me to choose quantum algorithms as my thesis topic was to get to know some of the problems for which there exists faster quantum algorithm than the best known classical method. I chose the topic of matrix inversion which is a central and deeply studied research area of recent mathematics.

The structure of my thesis is designed in a way that it starts by introducing frequently used and well-known concepts, and then culminates in two sophisticated methods that build upon the earlier concepts presented in the beginning. Throughout the work, I aimed to synthesize existing knowledge and provide a comprehensive analysis in the context of these methods.

In the *first chapter*, I provide the necessary background to understand the latter parts of the thesis. This part contains the relevant fundamental definitions, theorems and I touch upon the most common techniques of quantum computing many of which are vast research topics themselves. These are quantum encoding, Quantum Fourier Transform and Phase Estimation, Hamiltonian simulation and Amplitude Estimation. Besides, I also give a short overview of linear regression which is a classical supervised learning method of common occurrence, that will bear fruit in chapter three.

The *second chapter* demonstrates a pioneering quantum algorithm (by Harrow, Hassidim and Lloyd from 2008) for solving an $Ax = b$ linear systems of equations (with some restrictions). This is a crucial part of the thesis, not only because this task lies in the centre of many real-life problem, but also further quantum algorithms use this as a subroutine. The discovery of this algorithm has given birth to quantum machine learning, which is a brand new field in quantum computing. The core of this process is to calculate the inverse matrix. This way, applying A^{-1} to $|b\rangle$ yields $|x\rangle$ as the output. The steps of this algorithm will also serve as a template for other methods. The HHL-method has definite limitations that are also mentioned in order to cover the full picture.

The *third chapter* goes a little bit further. In this part, we focus on finding the best linear regression model, which involves computing the pseudo-inverse of a given matrix. We will provide two methods for this purpose. It is important to add that this task is more general than the previous one, as the pseudo-inverse of an invertible matrix coincides with its inverse. At first, I summarise the initial quantum algorithm to solve this problem, which stems from the authors of the previous algorithm. This, however, heavily builds on the HHL-algorithm, for which reason, I demonstrate another algorithm that utilizes more sophisticated methods.

When writing this thesis, I tried to focus my attention on the tasks in which quantum computers outperform classical ones. A few of the algorithms in this work are of this kind, however, we shouldn't ignore the fact, that they have their limitations. I also tried to highlight these when it was relevant.

1 Preliminaries

1.1 Quantum basics

In this subsection, I provide the absolute necessary background to the mathematics of the quantum world. I cover the basic notations, definitions and statements one needs to understand this thesis. For further details, see [18] or [14].

In quantum algorithms, we work with special N-dimensional vectors (so-called *quantum states*) which are conventionally written in the following form:

$$|\phi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + \ldots + \alpha_{N-1} |N-1\rangle ,$$

where $|j\rangle$ is the j^{th} basis vector in the N-dimensional space.

Its conjugate transpose is a row vector denoted by $\langle j| := \overline{|j\rangle}^T$.

Definition 1. *The vector $|\phi\rangle = \alpha |0\rangle + \beta |1\rangle \in \mathbb{C}^2$ is a **qubit** if $|\alpha|^2 + |\beta|^2 = 1$.*

Definition 2. *Vectors in the form of $\sum\limits_{j=0}^{N-1} \alpha_j |j\rangle \in \mathbb{C}^N$, where $N = 2^n$, are called **n-qubit registers** if $\sum\limits_{j=0}^{N-1} |\alpha_j|^2 = 1$. The complex numbers, α_j-s are called **amplitudes**.*

Transforming a register into another one is defined as a multiplication by a matrix of the respective size. The result of the multiplication should be a register as well, meaning that the matrix should preserve the norm. For this reason, only **unitary** matrices are allowed: for which $\|Ax\| = \|x\|$.

Property 1. *The following two conditions are equivalent:*

1. *A is unitary,*
2. *$A^* = A^{-1}$, where $A^* = \overline{A}^T$ per definition.* $\square$

Here, I list the most common unitary operators:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad R_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{1}$$

Matrix X swaps $|0\rangle$ and $|1\rangle$ that corresponds to the classical negation, matrix Z puts a negative sign in front of $|1\rangle$ and R_ϕ rotates the phase of the $|1\rangle$-state by an angle $|\phi\rangle$.

Another variation of R_ϕ is $R_s = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^s} \end{pmatrix}$. For $s = 1$, it is exactly Z, while for large s, R_s is close to I.

Other than applying operators, we can **measure** the register $\sum\limits_{j=0}^{N-1} \alpha_j |j\rangle \in \mathbb{C}^N$, which outputs $|k\rangle$ with probability $|\alpha_k|^2$. Note that this definition makes sense since $\sum\limits_{j=0}^{N-1} |\alpha_j|^2 = 1$ holds.

The most frequently used matrix H is the so-called *Hadamard*, which has the following special property:

$$H|j\rangle = \frac{1}{2}\big(|0\rangle + (-1)^j |1\rangle\big), \quad j \in \{0,1\}. \tag{2}$$

This implies that applying H to either $|0\rangle$ or $|1\rangle$ and measuring the resulting state, gives the output $|0\rangle$ or $|1\rangle$ with equal probability.

Applying more operations to one qubit can be done by multiplying the respective matrices.

Applying more operations to different qubits of a register simultaneously can be defined by tensor products. For example, $H|0\rangle \otimes H|0\rangle =: H^{\otimes 2}|00\rangle$. In this case $|00\rangle$ is equal to $|0\rangle \otimes |0\rangle$ by definition. This notation is used for other cases: let $b_i \in \{0,1\}$ for $i = 1, \dots, k$, then $|b_1, \dots, b_k\rangle := |b_1\rangle \otimes \dots \otimes |b_k\rangle$.

Hadamard matrix has the following ubiquitous property:

Property 2. $H^{\otimes n}|0^n\rangle = \frac{1}{\sqrt{2^n}} \sum\limits_{j \in \{0,1\}^n} |j\rangle.$ $\square$

Operators can be extended to affect on two or more qubits simultaneously. First, I list the most common 2-qubit gates.

A direct calculation shows that the following operator swaps the two qubits it acts on.

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Another ubiquitous one is *controlled-NOT*.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

It can be verified by a quick calculation, that $CNOT$ negates the second qubit of its input if the first qubit is $|1\rangle$, and behaves as identity if the first qubit is $|0\rangle$.

More generally, if U is a unitary 1-qubit operator: $U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$, then controlled-U is a 2-qubit operator defined as:

$$CU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{11} & u_{12} \\ 0 & 0 & u_{21} & u_{22} \end{pmatrix}.$$

A short calculation shows that CU applies U to the second qubit if the first qubit is $|1\rangle$ and behaves as identity if the first qubit is $|0\rangle$. The first qubit is called controlled-qubit, as it is left unchanged.

Other than one- or two-qubit operators, many-qubit operators play important role, too. An ubiquitous one is the so-called *controlled-rotation.* Let's say we have a number $0 \leq \phi < 1$, which has a d-bit representation: $\phi = \overline{0.\phi_1, \ldots, \phi_d}$, and it is stored in d qubits.

Then, there exists a $d+1$-qubit operation which rotates $|0\rangle$ in the following way:

$$|\phi\rangle |0\rangle \to |\phi\rangle \left(\sqrt{1 - \frac{C}{\phi^2}} |0\rangle + \frac{C}{\phi} |1\rangle \right), \tag{3}$$

where C is a normalizing constant. The exact implementation of this operation is not covered in this thesis, it can be found in [11].

Applying a function f in quantum computing is usually done in the following way: $|x\rangle |0\rangle \to |x\rangle |f(x)\rangle$. The same idea is used when reading an element of a matrix X and vector y. As we will be working with matrices and vectors, we assume that the following operators (so-called *queries* or *black-box operators*) exist for this purpose:

$$P_X\big(|j\rangle |k\rangle |z\rangle\big) = |j\rangle |k\rangle |z \oplus x_{j,k}\rangle,$$

and

$$P_y |j\rangle |z\rangle = |j\rangle |z \oplus y_j\rangle, \quad \text{for all } j \text{ and } k,$$

where $\oplus$ denotes the bitwise addition *modulo* 2. In both cases, the last register holds for a bit string representing the entry at the position given by the first (two) registers. Both P_X and P_y are unitary operators as they are permutation matrices.

In the algorithms presented in this thesis, we mostly work with multiple registers simultaneously. For clearer visibility, we use lower indices both for the registers and the operators that indicates which operator acts on which register. In the following example, the first register is $|\phi\rangle$, the second one is $|\psi\rangle$ and the third one is $|\nu\rangle$, while operator M_j $(j = 1, 2, 3)$ acts on the j^{th} register.

$$M_1 M_2 M_3 \Big(|\phi\rangle_1 |\psi\rangle_2 |\nu\rangle_3 \Big) = \big(M_1 |\phi\rangle_1\big) \otimes \big(M_2 |\psi\rangle_2\big) \otimes \big(M_3 |\nu\rangle_3\big).$$

This notation will prove exceptionally useful in Section 1.4 and Section 2.1 and Section 3.2.

Until now, we introduced several matrices (operations) which can be used easily in theory. However, building a quantum computer raises the questions which unitary operators to implement physically. According to the Solovay-Kitaev theorem ([14]), with the usage of H, $R_{\pi/4}$ and $CNOT$, any other unitary matrix can be arbitrarily well approximated.

In the remaining part of the thesis, we call the set of matrices I already introduced (with the exception of queries, P_X and P_y) ***elementary matrices***, meaning that we take their implementation for granted. When calculating the complexity of each algorithm, we will count the number of times they are utilized.

Definition 3. *The number of times elementary gates are used is called* ***gate complexity****. The number of the uses of P_X and P_y is called* ***query complexity****.*

I will often refer to gate complexity as *runtime*, *running time* or simply *complexity.*

Remark. *For an $N \times N$ matrix X and N-dimensional vector y, we assume that P_X and P_y can be implemented in $O\big(poly(\log N)\big)$ gate complexity.*

Remark. *All of the algorithms demonstrated in this thesis exclusively utilize elementary matrices and queries (P_X and P_y for the respective matrix X and vector y).*

The question naturally arises how to encode data in quantum states.

Method A is *digital encoding*: if we have a numerical value N in the decimal system whose binary representation is $y_1, \dots, y_n$, where y_js are bits, then it can be associated with the quantum state $|y_1\rangle \otimes \ldots \otimes |y_n\rangle$. For example, number 5 is represented by $|101\rangle$.

Method B is *analog encoding*: the N-dimensional data with complex entries $\underline{y} = (y_1, \dots, y_N)$ is represented by $\frac{1}{\sqrt{\sum_{j=0}^{N-1} y_j^2}} \sum_{j=0}^{N-1} y_j |j\rangle$. Here – on the contrary with digital encoding – the information is stored in the amplitudes.

Both of them have pros and cons: the first scheme is easier to execute but it can require too many number of qubits. Later, in Chapter 2, both methods will be utilized. In many steps of the algorithms, we will implicitly use that conversion between the two form is possible according to [13].

1.2 Quantum Fourier Transform

Fourier transform is a widely used tool in many branches of engineering, computational sciences and mathematics. Its faster version, the so-called Fast Fourier Transform was one of the most significant milestones of the 20th century mathematics, which provides a speed-up for polynomial multiplication, just to name its most well-known application.

However groundbreaking it was, its quantum counterpart ([7]) is even exponentially faster: Fast Fourier Transform takes $O(N \log(N))$ steps, while for Quantum Fourier Transform only $O(\log^2(N))$ gates are needed. What more interesting is the fact that Fourier transform is the only known tool in quantum computation which gives exponential advantage. The most significant case where Quantum Fourier Transform is used as a subroutine is the Quantum Phase Estimation, which is an algorithm for estimating the eigenvalues of a unitary matrix. This is covered in Section 1.3.

In this part, I demonstrate how the Quantum Fourier Transform is implemented using simple operations. Later in the thesis, this will pop up again in the linear equation solver algorithm (in Chapter 2) as a subroutine.

Here, the operation we refer to as Fourier transform, is a $2^n \times 2^n$-size unitary matrix, all of whose entries have the same magnitude. Let $N = 2^n$ and let F_N denote the matrix mentioned above. The (j, k)-entry of F_N is $\frac{1}{\sqrt{N}} e^{\frac{2\pi i \cdot jk}{N}}$.

As a quantum operation, it takes the following form:

$$F_N |k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi i \cdot jk}{N}} |j\rangle \in \mathbb{C}^N. \tag{4}$$

Here, $|k\rangle$ and $|j\rangle$ are basis vectors of $\mathbb{C}^N$, therefore it is more convenient to use their binary representation:

- $k = \overline{k_1, \ldots, k_n} = \sum_{r=1}^{n} k_r \cdot 2^{n-r}$,
- $j = \overline{j_1, \ldots, j_n} = \sum_{r=1}^{n} j_r \cdot 2^{n-r}$,

where each $k_r, j_r \in \{0, 1\}$.

The following lemma gives us an equivalent form of (4), which will suggest how the circuit implementation should look like.

Lemma 1.1. *[18] Let $N = 2^n$ and $|k\rangle = |k_1, \ldots, k_n\rangle$ be a basis vector in $\mathbb{C}^N$. Then,*

$$F_N |k\rangle = \bigotimes_{r=1}^{n} \frac{1}{\sqrt{2}} \Big(|0\rangle + e^{\frac{2\pi i k}{2^r}} |1\rangle \Big). \tag{5}$$

Proof. First, we reformulate the left-hand side of the equation.

$$F_N |k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi i jk}{N}} |j\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{N-1} e^{\frac{2\pi i jk}{2^n}} |j\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{N-1} e^{2\pi i k \left(\sum_{r=1}^{n} j_r 2^{-r} \right)} |j\rangle .$$

Here we only used the binary representation of $|j\rangle$ and the fact that $j/2^n = \sum_{r=1}^{n} j_r 2^{-r}$.

From now on, let's call

$$\frac{1}{\sqrt{2^n}} \sum_{j=0}^{N-1} e^{2\pi i k \left(\sum_{r=1}^{n} j_r 2^{-r} \right)} |j\rangle$$

left-hand side, and

$$\bigotimes_{r=1}^{n} \frac{1}{\sqrt{2}} \big(|0\rangle + e^{\frac{2\pi i k}{2^r}} |1\rangle \big)$$

right-hand side.

It is trivial to see that they are both vectors of dimension 2^n.

To prove their equality, let's fix an index $j' \in \{0, 1, \ldots, 2^n - 1\}$, whose binary representation is $\overline{j'_1, \ldots, j'_n}$. We claim that the j'-th coordinate of the left-hand side and the right-hand side are the same.

For clearer visibility, let us introduce a set of indices, $S = \{s \in \{1, \ldots, n\} : j'_s = 1\}$.

Equivalently, $j'_s = 1 \iff s \in S$.

The j'-th coordinate of the left-hand side is the coefficient of $|j'\rangle$, hence it is $\frac{1}{\sqrt{2^n}} e^{2\pi i k \left(\sum\limits_{r=1}^{n} j'_r 2^{-r} \right)}$. This is equal to $\frac{1}{\sqrt{2^n}} e^{2\pi i k \cdot \left(\sum\limits_{r \in S} j'_r 2^{-r} \right)}$, since if $j_r = 0$ for any r, then it doesn't contribute to the sum.

Using the definition of tensor product, the j'-th coordinate of the right-hand side is exactly $\frac{1}{\sqrt{2^n}} e^{2\pi i k \cdot \left(\sum\limits_{r \in S} j_r 2^{-r} \right)}$. □

Let us see how it can be implemented in the simplified case of $n = 4, N = 16$. We have $k \in \{0, 1, \ldots, 15\}$ whose binary form is $k = \overline{k_1, k_2, k_3, k_4}$.

In this case, substituting into equation (5) gives us the following form:

$$F_{16} |k_1, k_2, k_3, k_4\rangle =$$

$= \frac{1}{\sqrt{2}} \big(|0\rangle + e^{2\pi i \cdot 0.k_4} |1\rangle \big) \otimes \frac{1}{\sqrt{2}} \big(|0\rangle + e^{2\pi i \cdot 0.k_3 k_4} |1\rangle \big) \otimes \frac{1}{\sqrt{2}} \big(|0\rangle + e^{2\pi i \cdot 0.k_2 k_3 k_4} |1\rangle \big) \otimes \frac{1}{\sqrt{2}} \big(|0\rangle + e^{2\pi i \cdot 0.k_1 k_2 k_3 k_4} |1\rangle \big)$.

Here, we used that $e^{2\pi i} = 1$ and $j/2^n = \sum\limits_{r=1}^{n} j_r \cdot 2^{-r}$, which explains why the last r values of k appeared in the $n - r$ qubits.

To prepare F_{16}, we will use a 4-qubit quantum circuit, where the initial qubits are $|k_1\rangle, |k_2\rangle, |k_3\rangle$ and $|k_4\rangle$.

The easiest one to prepare is $\frac{1}{\sqrt{2}} \big(|0\rangle + e^{2\pi i \cdot 0.k_4} |1\rangle \big)$, as we only need to apply one Hadamard to $|k_4\rangle$, based on (2) and the fact that $e^{2\pi i \cdot 0.k_4} = (-1)^{k_4}$.

Similarly, applying Hadamard to $|k_3\rangle$ gives us $\frac{1}{\sqrt{2}} \big(|0\rangle + e^{2\pi i \cdot 0.k_3} |1\rangle \big)$. In order to set the amplitude of $|1\rangle$ to $e^{2\pi i \cdot 0.k_3 k_4}$, we have to multiply with $e^{2\pi i \cdot 0.0 k_4}$. Applying one controlled-R_2 does so, where the controlled qubit is $|k_4\rangle$. This should obviously come before applying Hadamard to the fourth qubit.

The remaining two qubits are prepared in an analogue way, the only difference is in the number controlled-R_s gates.

As a quantum circuit (for $n = 4$), it pictures as follows:

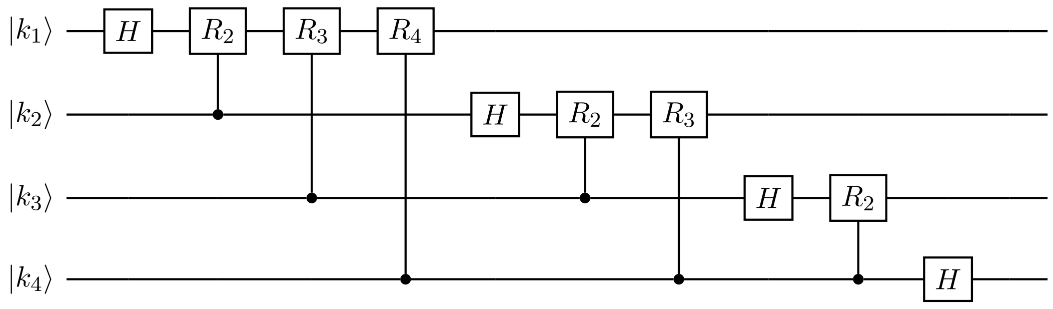

Figure 1. *Quantum circuit of* F_{16}

The general case (where n is arbitrary) is quite analogous to this one.

We use at most n gates for any of the n qubits, which implies that the number of gates is

$$O(n^2) = O(\log^2(N)).$$

It is important to add, that we can obtain the inverse Fourier transform (F_N^{-1}) by reversing the order of the gates and interchanging operation H with H^*.

For the next part, we need the following special property of F_N.

Remark. *On* $|0^n\rangle$*, the Fourier transform* F_N *has the same effect as* $H^{\otimes n}$*, where* $2^n = N$*:*

$$F_N |0^n\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle . \tag{6}$$

1.3 Quantum Phase Estimation

In this part, I will demonstrate the most common case, where Quantum Fourier Transform is used as a subroutine. The name of this method is Quantum Phase Estimation (hereafter referred to as QPE), which can be formulated as follows.

Given a unitary U and one of its $|\Phi\rangle$ eigenvector as a quantum state

$$U |\Phi\rangle = \lambda |\Phi\rangle , \tag{7}$$

the eigenvalue, λ, is to be found.

We don't assume to know Φ explicitly, all we need is to have it as a quantum state.

The fact the U is unitary implies that $|\lambda| = 1$, meaning that λ is on the unit circle. Formally, $\lambda = e^{2\pi i \phi}$ for some $\phi \in [0, 1)$.

Let n denote the number of bits which describe ϕ: $\overline{0.\phi_1, \ldots, \phi_n}$ and let's assume that no more bits are needed to accurately describe ϕ. Having fixed n, let $N := 2^n$.

The algorithm demonstrated below will calculate ϕ, after which calculating λ can be done using constant number of elementary gates.

The Phase Estimation algorithm consists of three major steps.
The initial state is $|0^n\rangle\,|\Phi\rangle$, to which we use the mapping $F_N \otimes I$. Because of (6), we obtain

$$\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle\,|\Phi\rangle\,.$$

After this, we apply the map

$$|j\rangle\,|\Phi\rangle \to |j\rangle\,U^j\,|\Phi\rangle = e^{2\pi i\phi j}\,|j\rangle\,|\Phi\rangle\,, \tag{8}$$

which gives us

$$\left(\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i\phi j}\,|j\rangle\right) \bigotimes |\Phi\rangle\,.$$

By (4), this is equal to $\Big(F_N\,|N\cdot\phi\rangle\Big)\,|\Phi\rangle = \Big(F_N\,|2^n\cdot\phi\rangle\Big)\,|\Phi\rangle\,.$
Now, we just apply the inverse Fourier transform F_N^{-1} to the first n qubits and obtain

$$F_N^{-1}F_N\,|2^n\phi\rangle\,|\Phi\rangle = |2^n\phi\rangle\,|\Phi\rangle = |\phi_1,\ldots,\phi_n\rangle\,|\Phi\rangle\,.$$

Theorem 1.2. *[11] The operation defined in Eq. (8) can be implemented using $O(2^n) = O(N)$ elementary gates.*

Corollary 1.2.1. *The runtime of QPE algorithm is $O(2^n) = O(N)$.*

Remark. *[11] Many times, one just needs the eigenvalue by n bit precision, where n is now an arbitrary integer. The above procedure can be modified such that it outputs an n-bit estimation of ϕ. The error of this modified algorithm cannot be more than 2^{-n}, trivially. It can be shown that the running time of this algorithm is also exponential in the number of the desired bit-precision, therefore $O(2^n)$.*

Alternatively, we could claim that the complexity of QPE is $O(1/\varepsilon)$, where ε denotes the desired precision.

The Quantum Phase Estimation turns up as a subroutine in many well-known algorithms. The pioneering integer factoring algorithm of Shor is a typical example, although that one is not covered in this thesis.

In Chapter 2, we will deal with system of linear equations, where, at some point, we will refer back to this subsection invoking QPE as a subroutine.

1.4 Hamiltonian simulation

Several times in this thesis, we want to utilize a Hermitian operator A (which means that $A^* = A$ holds), that is not necessarily unitary. A typical way to remedy this issue is to convert it to a unitary matrix

$$e^{iA} = \frac{(iA)^0}{0!} + \frac{iA}{1!} + \frac{(iA)^2}{2!} + \dots$$

and using this in the further steps. The main reason for doing so is that if $Au_j = \lambda_j u_j$, then $e^{iA}u_j = e^{i\lambda_j}u_j$. First, we have to make sure that the expression e^{iA} is unitary.

Lemma 1.3. *If $A \in \mathbb{C}^{N\times N}$ is a Hermitian matrix, then e^{iA} is unitary.*

Proof. All we have to check is $(e^{iA})^{-1} = (e^{iA})^*$.

We work in the eigenbasis of A, so that the matrix representation of e^{iA} is diagonal: if u_j denotes the j^{th} eigenvector of A with eigenvalue λ_j ($Au_j = \lambda_j u_j$), then

$$e^{iA} = \begin{bmatrix} e^{i\lambda_1} & & & \\ & e^{i\lambda_2} & & \\ & & \ddots & \\ & & & e^{i\lambda_N} \end{bmatrix}. \tag{9}$$

It is straightforward to verify that $(e^{iA})^* = e^{-iA}$:

$$(e^{iA})^* = \left(\sum_{n=0}^{\infty} \frac{(iA)^n}{n!}\right)^* = \sum_{n=0}^{\infty} \frac{(iA^*)^n}{n!} = \sum_{n=0}^{\infty} \frac{(iA)^n}{n!} = e^{-iA}, \tag{10}$$

where in the third equation, we exploited the Hermitian property of A.

Using (9) and (10), we obtain that

$$(e^{iA})^* = e^{-iA} = \begin{bmatrix} e^{-i\lambda_1} & & & \\ & e^{-i\lambda_2} & & \\ & & \ddots & \\ & & & e^{-i\lambda_N} \end{bmatrix}.$$

Consequently, $e^{iA}(e^{iA})^* = I = (e^{iA})^* e^{iA}$, which proves the statement. □

The conversion of $A \to e^{iA}$ is called *Hamiltonian simulation*, which we introduce in two special cases. But first, we define a matrix property, which will play a crucial role further on.

Definition 4. *Matrix M is **s-sparse**, if it has at most s nonzero elements in each row, where s is a constant, independent of the size of M. We also use the term 'sparse' and '$O(1)$-sparse'.*

Remark. *When working with sparse matrices, we always suppose that there exists a polynomial time classical algorithm which, given a row index $r \in \{1, \dots, N\}$, outputs the non-zero entries in row r.*

Theorem 1.4. *([12]) Let A be an s-sparse Hermitian matrix of size $N \times N$. Then, simulating e^{-iAt} can be done with a quantum algorithm in time $O(\log(N)s^2t)$.*

We will make use of this theorem in the linear equation system solver algorithm, while the next theorem will pop up in the quantum linear regression model.

Theorem 1.5. *([12]) Let $H \in \mathbb{C}^{N\times N}$ be a Hermitian operator in the following special form:*
$H = \big(\langle G| \otimes I\big)U\big(|G\rangle \otimes I\big)$, where $U \in \mathbb{C}^{Nd\times Nd}$ is a unitary and $|G\rangle = \hat{G}\,|0\rangle \in \mathbb{C}^d$.

Then, there exists an algorithm that simulates e^{-iHt} with precision ε and failure probability $O(\varepsilon)$ by making $O(t + \log(1/\varepsilon))$ uses of controlled-$\hat{G}$ and controlled-U.

Corollary 1.5.1. *Let X be a matrix of size $N \times d$, whose singular values lie in $[1/\kappa, 1]$ and let matrix A be defined as*

$$A := |0\rangle\langle 1| \otimes X + |1\rangle\langle 0| \otimes X^T = \begin{pmatrix} 0 & X^T \\ X & 0 \end{pmatrix}.$$

Then, there exists an algorithm that simulates e^{-iAt} with precision ε and failure probability $O(\varepsilon)$ by making $O\Bigg(d\big(\sqrt{d}t + \log\big(\tfrac{1}{\varepsilon}\big)\big)\Bigg)$ uses of P_X.

Proof. All we have to do is to construct matrices G and U such that $A = \big(\langle G| \otimes I\big)U\big(|G\rangle \otimes I\big)$.

At first, let's define $\sigma = \sigma(X) := \dfrac{\max_{1\le j\le N}\|x_j\|^2}{\sqrt{\frac{1}{N}\sum\limits_{j=1}^{N}\|x_j\|^2}}$, where x_j is the j^{th} row of X and $\|.\|$ denotes the conventional ℓ_2-norm.

Next, we define matrix $\tilde{X} := X \cdot \dfrac{\sqrt{N}}{\sigma\sqrt{d}}$, where. This will make sense in equation (13), when V is introduced.

Since all $s_j(X)$ singular value of X is at most 1, we obtain that

$$tr(X^TX) = \sum_{j=1}^{d} s_j^2(X) \le d.$$

Hence,

$$\max_{1\le j\le N}\|x_j\| \le \frac{\sigma \cdot \sqrt{tr(X^TX)}}{\sqrt{N}} \le \frac{\sigma\sqrt{d}}{\sqrt{N}}. \tag{11}$$

From equation (11), we get that:

$$\max_{1\le j\le N}\|\tilde{x}_j\| \le 1. \tag{12}$$

Next, we construct the matrices that give us the desired mapping.

Let V be the operator which executes the following mapping:

$$V : \big(|0, i\rangle_1 \big) \otimes \big(|0, 0^m\rangle_2 \, |0\rangle_3 \big) \to |0, i\rangle_1 \otimes \left(\sum_{j=1}^{d} \tilde{x}_{i,j} \, |1, i\rangle_2 \, |0\rangle_3 + \sqrt{1 - \|\tilde{x_i}\|^2} \, |1, 1\rangle_2 \, |1\rangle_3 \right), \quad (13)$$

for $1 \leq i \leq N$,

and

$$V : \big(|1, j\rangle_1 \big) \otimes \big(|0, 0^m\rangle_2 \, |0\rangle_3 \big) \to |1, j\rangle_1 \otimes \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |0, i\rangle_2 \, |0\rangle_3 \,, \quad (14)$$

for $1 \leq j \leq d$ and $m = \Theta(\log(N))$.

Note that in equation (13) we have a non-negative number under the square root due to (12).

The mapping V makes it possible to read the elements of X. In equation (13), a row index can be set in the second part of the first register. This way, we gain access to the chosen row of X through the amplitudes. While in equation (14), column index is free of choice.

With these operators in our pocket, let's define operator U

$$U = V^* \cdot (SWAP_{1,2} \otimes I_3\big) \cdot V,$$

and $|G\rangle$ as follows:

$$|G\rangle = |0, 0^m\rangle_2 \, |0\rangle_3 \,.$$

Here $\hat{G}$ is the identity.

Going through the calculations, we obtain that

$$\frac{1}{\sigma\sqrt{d}} \begin{pmatrix} 0 & X \\ X^T & 0 \end{pmatrix} = \big(\langle G| \otimes I \big) V^* (SWAP_{1,2} \otimes I_3) V \big(|G\rangle \otimes I \big).$$

Equivalently, $A = \sigma\sqrt{d} \cdot H$, meaning that we reduced the problem to Thm 1.5.

What remained is to make sure that the whole process can be achieved using the query P_x no more than $O(d)$ times.

We can clarify that $SWAP$ can be implemented in time *poly*$(\log(N))$. The mapping in (14) can be implemented in $O(\log(N))$ using Hadamard.

We only need to construct the mapping in (13) by using P_X no more than $O(d)$ times.

To do this, we first accomplish the transformation

$$|0, i\rangle_1 \, |0, 0^m\rangle_2 \, |0\rangle_3 \to |0, i\rangle_1 \, |0, 0^m\rangle_2 \, |0\rangle_3 \left(\bigotimes_{j=1}^{d} |x_{i,j}\rangle \right)_4 .$$

This requires nothing else than $O(d)$ uses of P_X, as we just query all the d element of the i^{th} row of X.

With the elements of X in the fourth register, we can get the state:

$$|0,i\rangle_1 \left(\sum_{j=1}^{d} \tilde{x}_{i,j} |1,j\rangle_2 |0\rangle_3 + \sqrt{1 - \|\tilde{x_i}\|^2} |1,1\rangle_2 |1\rangle_3 \right) \left(\bigotimes_{j=1}^{d} |x_{i,j}\rangle \right)_4 .$$

We won't delve into the details of this step, but according to ([15]), it can be done in $O(d \log(N))$. Setting back the last register to $|0\rangle$, leaves us with the desired state.

The fact that the whole procedure requires no more than $O(d)$ uses of P_X implies that the whole algorithm fulfils all the requirements we demanded.

□

1.5 Amplitude Estimation

In this section, we demonstrate a well-known technique – based on articles on [4] and [11] – to prepare a quantum state from an amplitude. This algorithm will be mostly invoked as a subroutine in Chapter 3.2, where the amplitudes will store the information to be outputted.

Formally, we have the state

$$|y\rangle = \sqrt{p} |y_{good}\rangle + \sqrt{1-p} |y_{bad}\rangle ,$$

where $p << 1$ and $|y_{good}\rangle$ is orthogonal to $|y_{bad}\rangle$. The aim is to construct an algorithm that estimates p and outputs it in a quantum state. But instead of $|p\rangle$, the Amplitude Estimation algorithm will output $\phi \in [0, \pi]$, for which $p = \sin(\phi/2)$. It is equivalent to say, that the angle of $|y\rangle$ and $|y_{good}\rangle$ is $\phi/2$.

Here, we assume that we have access to two matrices. The first one, S_y, is the reflection to the hyperplane orthogonal to $|y\rangle$. The second one, S_{good} is the reflection to the hyperplane orthogonal to $|y_{good}\rangle$, or equivalently, it reflects on $|y_{bad}\rangle$. Normally, the existence of S_{good} is the stronger assumption.

Hence, they are defined this way and also called Householder transformation.

$$S_y = I - 2 |y\rangle \langle y| ,$$

and

$$S_{good} = I - 2 |y_{good}\rangle \langle y_{good}| .$$

Let G denote these two reflections after each other:

$$G = S_y S_{good}.$$

A straightforward calculations shows that in the orthonormal basis of $\{y_{good}, y_{bad}\}$ the matrix representation is

$$S_{good} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} .$$

Similarly,

$$S_y = \begin{pmatrix} -\cos(\phi) & \sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix} .$$

Therefore, we obtain the following form of G:

$$G = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix},$$

which is a rotation matrix.

Its two eigenvectors as quantum states are

$$|y_\pm\rangle = \frac{1}{\sqrt{2}}\big(|y_{good}\rangle \pm i\,|y_{bad}\rangle \big),$$

with eigenvalues $e^{\pm i\phi}$. All we have to do is to apply Quantum Phase Estimation to get the state $e^{\pm i\phi}$.

Therefore, the total runtime of Amplitude Estimation is dependent of the precision with which the Quantum Phase Estimation is invoked.

1.6 Classical linear regression

Given a data set: $\big(x_j, y_j\big)_{j=1}^{N}$, the *inputs*, x_j are elements of $\mathbb{R}^d$, the *outputs*, y_j are elements of $\mathbb{R}$. The vectors of inputs can be written in a matrix form: $X = ((x))_{i,j} \in \mathbb{R}^{N\times d}$, where the j^{th} row of X is x_j. For simplicity, we will refer to X as 'data matrix'.

We are searching for a linear model in a given d-dimensional vector space, e.g. a vector $\beta = (\beta_1, \ldots, \beta_d)$ such that $\beta_1 x_{i,1} + \ldots + \beta_d x_{i,d} \approx y_i, \quad \forall i = 1, \ldots, N$.

In a matrix form, this can be written as:

$$y \approx X\beta.$$

Here we only consider the case when X has full rank.

First, assume that $N > d$, which means that we've got more data points than the dimension of the vector space. This implies that the linear system of equations is overdetermined, meaning that in a typical case there is no unique solution. A typical approach is to find $\hat{\beta}$ such that

$$\hat{\beta} = \arg\min_{\beta} \|X\beta - y\|^2, \tag{15}$$

which is called the *least squares method.* It is well-known that the solution of (15) is unique and has the following closed form:

$$\hat{\beta} = (X^T X)^{-1} X^T y.$$

We can introduce the notation $X^+ := (X^T X)^{-1} X^T$ which is called the *pseudo-inverse* of X. The $\hat{\beta}$ also has a nice geometrical interpretation: the closest point to y from the subspace spanned by the columns of X is $X\hat{\beta}$.

Computing $\hat{\beta}$ classically is normally done using QR-factorization in the following way. If $X = QR$, where Q has orthonormal columns and R is an upper triangular with non-zero diagonals, then $X^+ = R^{-1}Q^T$ and consequently $\hat{\beta} = R^{-1}Q^T y$. This way, we can avoid matrix inversion by backward substitution. The bad news is, this approach takes $O(Nd^2)$ steps.

In many examples, the input-matrix X is complex valued. The following general definition of pseudo-inverse covers this case as well.

Definition 5. $X^+ \doteq (X^*X)^{-1}X^*$.

The second case is when $N < d$. If X is assumed to be full rank, then the solution of $y = X\beta$ is not unique. Here, the so-called *least norm* approach is used: we search for a $\hat{\beta}$ which minimizes $\frac{1}{2}\|\beta\|^2$, with the condition of $y = X\beta$. In the quantum regression part of this thesis, I only care about the first case, when $N > d$. However, it is important to add, that the solution of these two cases show some similarities. In case of $N < d$, the sought-after $\hat{\beta}$ has the following closed form: $\hat{\beta} = X^T(XX^T)^{-1}y$, and the definition of pseudo-inverse could easily be extended to this case.

2 Linear system of equations

In this chapter, I mostly rely on the seminal article from 2008 ([10]) written by Harrow, Hassidim and Lloyd, but I also make use of a lecture note ([8]) written by Sevag Gharibian.

This chapter consists of three parts: in Section 2.1, I present a well-known algorithm for solving linear system of equation. Then, in Section 2.2, I compute its running time and demonstrate a theorem which pinpoints why this problem is so crucial. In Section 2.3, I show how it can be applied to solve linear differential equations.

This, the so-called HHL algorithm, has given rise to the field of quantum machine learning which is still in its infancy. Several quantum algorithms use this as a subroutine, therefore its theoretical significance is unquestionable.

However meaningful it seems, it certainly has some deficiencies as Scott Aaronson pointed out in his article ([1]). These eye-catching flaws of HHL algorithm (see below) raise the question whether it can be used in any practical case or this whole method is exclusively of theoretical significance. At the present time, we can conclude that the latter is the case. However, there is a high chance that it will find practical uses in the long-term future.

2.1 HHL algorithm for matrix inversion

As an input, we are given a matrix $A \in \mathbb{C}^{N \times N}$ and a vector $b \in \mathbb{C}^N$, the task is to find an $x \in \mathbb{C}^N$ such that $Ax = b$.

The procedure, discussed below, solves this problem in *logarithmic* time in N, which is an exponential speed-up over any classical algorithm solving the same problem. However, HHL-algorithm doesn't provide an improvement in every practical cases. One of its downsides is that it outputs a quantum state which is always a vector of length 1 in $\mathbb{C}^N$, therefore it is only proportionate to $|x\rangle$. Also, it is not possible to query its elements, since a measurement spoils the state leaving us with a basis vector. In those cases where we need $|x\rangle$ just to perform further operations on it, the algorithm (described in this section) can prove useful, indeed.

We will solve the system of equation in a specific way. The core of the method below is the invertion of matrix A. With this in mind, solving the system reduces to applying the inverse matrix to the input b, outputting the solution vector x in a quantum state.

Notice that, inverting a matrix with a classical algorithm takes as much time as solving the system of equation. As pointed out in Section 1.2, Quantum Fourier Transform is exponentially faster than its classical counterpart. This is the major reason behind the fact that HHL-algorithm will provide an exponential acceleration to the best classical algorithm for solving linear system of equations.

As we are talking about quantum algorithms, our current state is always an element of $\mathbb{C}^{2^k}$ for a fixed $k \in \mathbb{N}$. For this reason, we dispense from the normalizing factor: all we care about is the angle of the given vector.

Definition 6. *We say that vector $v \in \mathbb{C}^{2^k}$ and vector $w \in \mathbb{C}^{2^k}$ are proportional, if there exists a constant $c \in \mathbb{R} - \{0\}$ such that $v = cw$, meaning that the direction of v and w are identical. For simplicity, I use the following notation: $v \sim w$. This relation is trivially symmetrical.*

Before moving on to the algoritms, let's begin with the **assumptions** we need.

- First, A is full rank. This implicates the invertibility of A, which assumption cannot be avoided.
- Without loss of generality, we can assume that A is a square matrix and Hermitian: $A = A^*$. If this isn't the case, solving
$$\tilde{A}\tilde{x} = \tilde{b}, \tag{16}$$
where $\tilde{A} = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix}$, $\tilde{x} = \begin{pmatrix} 0 \\ x \end{pmatrix}$, and $\tilde{b} = \begin{pmatrix} b \\ 0 \end{pmatrix}$ presents the solution of the original system of equations.
- We also suppose that:
  1. either b, as an input, is given in a quantum state: $|b\rangle$,
  2. or the assignment $b \to |b\rangle$ can be made efficiently.
- Normally, the speed of the linear algebraic algorithms depends on the properties of the input matrix. Here, we assume that $\|A\|_\infty = 1$, hence $\|A^{-1}\|_\infty = \kappa(A)$. This implies that $\lambda_{max}(A) = 1$ and $\lambda_{min}(A) = \frac{1}{\kappa(A)}$. This is equivalent to saying that A is *well-conditioned.*
- The sparseness of A is also supposed, which is by far the most severe restriction of all.
- Furthermore, we also assume that the eigenvalues λ_j of matrix A require no more than $O(\log\log(N))$ bits to represent.

Another downside of the algorithm – as I pointed out earlier – that it outputs the quantum state $|x\rangle$ of $\log_2(N)$ qubits. This can be useful, when for example we want to apply further operations to this state or reveal statistical information about x, such as approximating the value of an inner product $\langle x|z\rangle$, where z is some fixed vector.

As you can see, HHL-algorithm does not exactly solves $Ax = b$ in general, but $Ax = b$ with the above conditions. On the top of that, it is also questionable whether it is possible to give a classical algorithm for solving exactly the same problem with similar complexity. At the present time, this is an open question. Having so many conditions raises the question if this algorithm has any practical use. However, the only severe condition listed above is the sparseness of matrix A. So far, the only known start-to-finish application of this algorithm is the speed-up of the calculation of electromagnetic scattering cross-sections ([6]).

With these assumptions in hand, let's delve into the details of the **algorithm**.
Since A is Hermitian, it can be written in a spectral form:
$$A = \sum_{j=1}^{N} \lambda_j |u_j\rangle \langle u_j|, \tag{17}$$
where $A|u_j\rangle = \lambda_j |u_j\rangle$. The eigenvalues, λ_j-s are real and the eigenvectors, the $|u_j\rangle$ form an orthonormal basis in $\mathbb{C}^N$, therefore $|b\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle$ with some coefficients β_j. The inverse matrix

can also be expressed in a similar form:

$$A^{-1} = \sum_{j=1}^{N} \frac{1}{\lambda_j} \left|u_j\right\rangle \left\langle u_j\right| . \tag{18}$$

Using these and the orthogonality of the eigenvectors, we get the following closed form of the wanted output:

$$\left|x\right\rangle = A^{-1}\left|b\right\rangle = \sum_{j=1}^{N} \frac{1}{\lambda_j}\beta_j \left|u_j\right\rangle . \tag{19}$$

In order to get this state as an output, the key is to smuggle in the eigenvalues of A. For this purpose, we will invoke the Quantum Phase Estimation algorithm described in Section 1.2.

First of all, A is not necessarily unitary, therefore applying the Hamiltonian simulation (Thm. 1.5) for A, we obtain $e^{iA} = \frac{(iA)^0}{0!} + \frac{iA}{1!} + \frac{(iA)^2}{2!} + \ldots$, which is unitary according to Lemma 1.3. For simplicity, we assume that this could be done efficiently.

Notice that $Au_j = \lambda_j u_j \Rightarrow e^{iA}u_j = e^{i\lambda_j}u_j$. Since $|e^{i\lambda_j}| = 1$, we get that $i\lambda_j = 2\pi i\phi$ for some $\phi \in [0,1)$, which is equivalent to $\lambda_j = 2\pi\phi$.

The algorithm is executed on three registers. Initially, $\left|b\right\rangle$ is stored in the first one, the second one is $\left|0^m\right\rangle$, where $m = O(\log(N))$ while the third one is a single qubit set to $\left|0\right\rangle$.

Since, the $\left|u_j\right\rangle$ form a basis, the quantum state $\left|b\right\rangle$ is equal to $\sum_{j=1}^{N} \beta_j \left|u_j\right\rangle$ for some unknown β_j-s. Formally, this means that the starting state is

$$\sum_{j=1}^{N} \beta_j \left|u_j\right\rangle_1 \left|0^m\right\rangle_2 \left|0\right\rangle_3 .$$

In the first step, we apply the Quantum Phase Estimation for unitary e^{iA}. After this, the current state is

$$\sum_{j=1}^{N} \beta_j \left|u_j\right\rangle_1 \left|\lambda_j\right\rangle_2 \left|0\right\rangle_3 .$$

Now that we have λ_j-s encoded in a quantum state, we use controlled-rotation (defined in eq. (3)) to the second and third register:

$$\left|\lambda_j\right\rangle_2 \left|0\right\rangle_3 \rightarrow \left|\lambda_j\right\rangle_2 \left(\sqrt{1 - \frac{1}{\lambda_j^2 \kappa^2(A)}}\left|0\right\rangle + \frac{1}{\lambda_j \kappa(A)}\left|1\right\rangle\right)_3 . \tag{20}$$

Having executed the controlled-rotation, we have the state

$$\sum_{j=1}^{N} \beta_j \left|u_j\right\rangle_1 \left|\lambda_j\right\rangle_2 \left(\sqrt{1 - \frac{1}{\lambda_j^2 \kappa^2(A)}}\left|0\right\rangle + \frac{1}{\lambda_j \kappa(A)}\left|1\right\rangle\right)_3 . \tag{21}$$

The constant factor $\kappa(A)$ is only needed so that we have a positive number under the square root.

In order to eliminate the eigenvalues, we apply the inverse of Quantum Phase Estimation to the first two registers as the next step of the procedure. At this time, we are in the state of

$$\sum_{j=1}^{N} \beta_j \left|u_j\right\rangle_1 \left|0^m\right\rangle_2 \left(\sqrt{1 - \frac{1}{\lambda_j^2 \kappa^2(A)}} \left|0\right\rangle + \frac{1}{\lambda_j \kappa(A)} \left|1\right\rangle \right)_3 .$$

All we have to do is to measure the third register in a standard basis. If we get $\left|0\right\rangle$ after the measurement, we repeat the whole algorithm, until the measurement outputs $\left|1\right\rangle$ in the third register. After that, we 'remove the garbage', e.g. apply operator X (defined in eq. (1)) to the third register which resets $\left|1\right\rangle$ to $\left|0\right\rangle$.

To sum up, the algorithm, as a flow diagram, looks as follows:

$$\left|b\right\rangle_1 \left|0^m\right\rangle_2 \left|0\right\rangle_3 = \sum_{j=1}^{N} \beta_j \left|u_j\right\rangle_1 \left|0^m\right\rangle_2 \left|0\right\rangle_3 \xrightarrow{(1)} \sum_{j=1}^{N} \beta_j \left|u_j\right\rangle_1 \left|\lambda_j\right\rangle_2 \left|0\right\rangle_3 \xrightarrow{(2)}$$

$$\xrightarrow{(2)} \sum_{j=1}^{N} \beta_j \left|u_j\right\rangle_1 \left|\lambda_j\right\rangle_2 \left(\sqrt{1 - \frac{1}{\lambda_j^2 \kappa^2(A)}} \left|0\right\rangle + \frac{1}{\lambda_j \kappa(A)} \left|1\right\rangle \right)_3 \xrightarrow{(3)}$$

$$\xrightarrow{(3)} \sum_{j=1}^{N} \beta_j \left|u_j\right\rangle_1 \left|0^m\right\rangle_2 \left(\sqrt{1 - \frac{1}{\lambda_j^2 \kappa^2(A)}} \left|0\right\rangle + \frac{1}{\lambda_j \kappa(A)} \left|1\right\rangle \right)_3 \xrightarrow{(4)}$$

$$\xrightarrow{(4)} \sum_{j=1}^{N} \beta_j \frac{1}{\lambda_j \kappa(A)} \left|u_j\right\rangle_1 \left|0^m\right\rangle_2 \left|0\right\rangle_3 \quad \sim \quad \sum_{j=1}^{N} \frac{1}{\lambda_j} \beta_j \left|u_j\right\rangle_1 \left|0^m\right\rangle_2 \left|0\right\rangle_3 = \left|x\right\rangle_1 \left|0^m\right\rangle_2 \left|0\right\rangle_3 .$$

For further applications, it's crucial to note that we can alter step (2) by setting the amplitude of $\left|1\right\rangle$ (and of $\left|0\right\rangle$, evidently) to other values. Let's call this step 'eigenvalue transformation'. With this degree of freedom, we are armed with a general template to produce the powers of A, not just its inverse.

In the figure below, I picture this general algorithm as a quantum circuit. The abbreviations QPE and EVT stand for Quantum Phase Estimation and 'eigenvalue transformation', respectively.

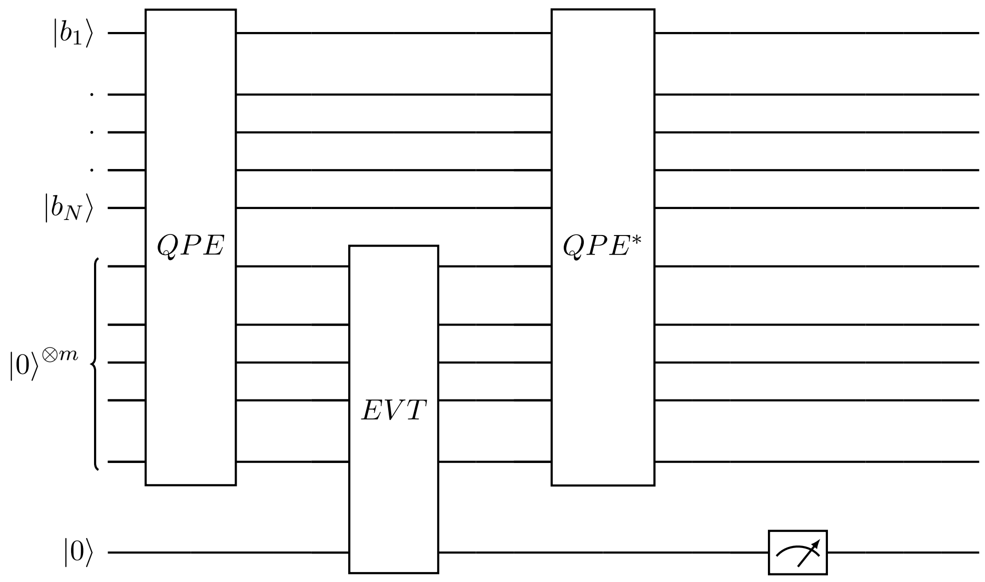

Figure 2. *Quantum circuit for generalized HHL*

2.2 Runtime analysis and BQP-completeness

From the description of the algorithm above, we can conclude the following:

Corollary 2.0.1. *The runtime of the HHL algorithm is*

$$O\bigg(\kappa^2(A)\cdot\big(T_b+\log N\big)\bigg), \tag{22}$$

where T_b denotes the number of gates required for constructing $|b\rangle$.

Proof. Measuring the third register gives us $|1\rangle$ with probability $\kappa^2(A)$. Hence, the expected number of times we have to repeat the whole algorithm is $\dfrac{1}{1/\kappa^2(A)} = \kappa^2(A)$.

One running of the algorithm consists of two phases, indeed. First, we have to produce $|b\rangle$, which takes T_b time. In the remaining part of the algorithm, the major subroutine is the sparse Hamiltonian simulation, as the Quantum Phase Estimation takes $O(\log(N))$ runtime, as we assumed that the eigenvalues need only $O(\log\log(N))$ bits to describe. According to Thm.1.4, Hamiltonian simulation in this case takes $O(ts^2\log(N))$ time. Considering that we applied it with $t=1$ and the fact that matrix A was sparse, we get the desired result. □

Remark. *If we demand $\kappa(A), T_b \in polylog(N)$, then the complexity of the HHL-algorithm is $O(poly(log(N)))$. If, furthermore, A happens to be unitary, then $\kappa(A)=1$ which reduces the complexity even more.*

Remark. *One of our assumption was that all eigenvalues can be written by $O(\log\log N)$ bits. There exists a modification of the aforementioned algorithm that does not rely on this assumption and provides an estimation for the output vector, $|x\rangle$. However, I did not address this aspect in my thesis.*

In the remaining part of this chapter, we prove that no significant improvement of the running time is possible. In particular, any quantum computation can be encoded into an instance of solving linear equations. For this purpose, we introduce some definitions and notations in the first step.

Let **P** denote the set of problems that can be solved by classical deterministic computers using polynomial time. Interchanging 'polynomial time' with 'polynomial space', we get the definition of **PSPACE**. Another well-known classical complexity class is **BPP**. It denotes the problems that can be solved by classical randomized computers using polynomial time with error probability $\leq 1/3$ on every input.

Its quantum counterpart is **BQP** (defined below) which can be thought of as the set of problems that can be efficiently solved by quantum computers.

Definition 7. *Let **BQP** denote the set of problems that can be solved by quantum circuits using polynomial time with error probability $\leq 1/3$ on every input. A problem is called **BQP**-complete, if any other problem of **BQP** can be reduced to it in polynomial time.*

The following series of containment holds.

Theorem 2.1. *[18] **P** $\subseteq$ **BPP** $\subseteq$ **BQP** $\subseteq$ **PSPACE**.*

Only the latter containment is not trivial. It is also important to note that equation between any two of these four complexity classes is not proved.

At this point, we must clarify that the proof of optimality will not be entirely precise in a mathematical sense. We rely upon the conjecture that **BQP** $\neq$ **PSPACE**, which is generally believed by mathematicians.

Having completed the preparations, we can now define the exact problem.

Definition 8. *The so-called **matrix inversion (MI)** problem is the following:*

The input is an invertible, Hermitian $A \in \mathbb{C}^{N\times N}$ matrix, for which $\frac{1}{\kappa(A)} \leq \lambda(A) \leq 1$ for any $\lambda(A)$ eigenvalue of A. We also assume $\kappa(A)$ to be in $poly(\log(N))$.

Let $|x'\rangle \sim A^{-1}\left|0^N\right\rangle$ be a unit vector and let $\Pi = |1\rangle\langle 1| \in \mathbb{C}^2$.

If $\langle x|\Pi\otimes I|x\rangle \geq \frac{2}{3}$, the output should be YES, if $\langle x|\Pi\otimes I|x\rangle \leq \frac{1}{3}$, the output should be NO, and the output can be an arbitrary YES or NO otherwise.

Theorem 2.2. *[10] **MI** is **BQP**-complete.*

Proof. The HHL-algorithm demonstrated in Section 2.1 proves the containment in **BQP**, thus only the **BQP**-completeness remains to be shown. The proof goes by contradiction: the existence of a quicker quantum algorithm consequences that **BQP** = **PSPACE**.

Let $V = V_m \cdot \ldots \cdot V_1$ be an arbitrary circuit in **BQP**, which acts on n qubits, where $m \in poly(n)$ and $n \in \log(N)$. We also assume that each V_j acts on 2 qubits, which fact doesn't spoil the polynomial time.

Our aim is to show that with the inversion procedure in hand, we can recognize any language that belongs to **BQP**. Thus, we want to map V to A, where the inverse of A behaves the same way as V. For this purpose, we introduce a new operator U.

$$\begin{aligned} U := |1\rangle\langle 0| \otimes V_1 + |2\rangle\langle 1| \otimes V_2 + \ldots + |m\rangle\langle m-1| \otimes V_m + \\ + |m+1\rangle\langle m| \otimes V_m^\star + |m+2\rangle\langle m+1| \otimes V_{m-1}^\star + \ldots + |2m-1\rangle\langle 2m-2| \otimes V_2^\star + \\ + |0\rangle\langle 2m-1| \otimes V_1^\star. \end{aligned} \tag{23}$$

Then, U is a block matrix that looks like this:

$$U = \begin{bmatrix} &&&&&&&& V_1^* \\ V_1 &&&&&&&& \\ & V_2 &&&&&&& \\ && \ddots &&&&&& \\ &&& V_m &&&&& \\ &&&& V_m^* &&&& \\ &&&&& V_{m-1}^* &&& \\ &&&&&& \ddots && \\ &&&&&&& V_2^* & \end{bmatrix}$$

We can notice some important properties of U.

First of all, in the main diagonal of UU^*, we have either $V_j V_j^*$ or $V_j^* V_j$. Consequently, $UU^* = I$, which means that U is unitary.

Moreover, U has another important feature:

- $\forall\ 0 \le j \le m$: $U^j \left|0^{\log m}\right\rangle |0^n\rangle = |j\rangle V_j \ldots V_1 |0^n\rangle$
- $\forall\ m+1 \le j \le 2m$: $U^j |0\rangle |0\rangle = |j\rangle V_{2m-j} \cdot \ldots \cdot V_1 |0\rangle$,

which follows directly from the definition of U.

In the next step of the proof, we introduce a new matrix:

$$A := I - U.$$

The inverse of A can be expressed as: $A^{-1} = \sum\limits_{l=0}^{\infty} U^l$. Therefore, we get:

$$|x\rangle \sim A^{-1} \left|0^{(\log m)+n}\right\rangle = \left(\sum_{l=0}^{\infty} U^l\right) |0\rangle \sim |0\rangle |0^n\rangle + |1\rangle V_1 |0^n\rangle + \dots |m\rangle V_m \cdot \dots \cdot V_1 |0^n\rangle .$$

Since $|j\rangle V_j \cdot \dots \cdot V_1 |0^n\rangle$ is a unit vector, the above expression can be simplified to:

$$|x\rangle = \frac{1}{\sqrt{m+1}}\big(|0\rangle |0^n\rangle + |1\rangle V_1 |0^n\rangle + \dots + |m\rangle V_m \cdot \dots \cdot V_1 |0^n\rangle \big).$$

Measuring the first register, we get $|m\rangle V |0^n\rangle$ with probability

$$\frac{1}{m+1}. \tag{24}$$

Using (24) and the assumption that V is in **BQP**, the following two statements hold.

If V accepts, then

$$\langle x|\Pi \otimes I|x\rangle \geq \frac{2}{3}\frac{1}{m+1}. \tag{25}$$

If V rejects, then

$$\langle x|\Pi \otimes I|x\rangle \leq \frac{1}{3}\frac{1}{m+1}. \tag{26}$$

Here, the notation Π stands for $|1\rangle \langle 1| \in \mathbb{C}^2$.

Our goal is to get rid of the $\frac{1}{m+1}$ factor in these bounds in order to have probability bounds 2/3 and 1/3. This can be done using the error reduction for **BQP**.

Furthermore, we should match the remaining assumptions of MI-problem.

First of all, U is $O(1)$-sparse, based on the definition of U and the assumption that each gate V_j acts on 2 qubits. This property is inherited to A, obviously.

In addition, we should make sure that A is an invertible, Hermitian matrix, with all of its eigenvalues lying in the interval of $[1/\kappa(A), 1]$ and $\kappa(A)$ is polylogarithmic in N. In order to meet these requirements, we modify the definition of A while still satisfying the properties we already met.

In the first try, we can choose A to be

$$A = I - e^{-1/m}U,$$

for scalar $e^{-1/m}$. For this definition, $\kappa(A) \in O(m)$. The reason behind this is that U is unitary and the fact that for normal matrices, the eigenvalues are the singular values in absolute value. Furthermore, A is now invertible.

This is still not the final form of A: we need to make it Hermitian as well. For this purpose, we set A to the following:

$$A = \begin{pmatrix} 0 & I - e^{-1/m}U \\ I - e^{-1/m}U^* & 0 \end{pmatrix}.$$

This way, A is trivially Hermitian, has a condition number of $O(m)$, the eigenvalues still lie in the desired interval while A remained to be invertible and sparse.

□

2.3 Solving differential equations

A very useful application of the HHL-algorithm took place in solving first-order ordinary linear differential equations ([3]):

$$x'(t) = A(t)x(t) + b(t),$$

where $A(t) \in \mathbb{C}^{N\times N}, b(t)$ and $x(t) \in \mathbb{C}^N$ and $t \in [0, T]$ for some $T \in \mathbb{R}$. The most typical way to solve this is discretization, using Euler-method on a uniform grid: $\Delta := T/N$, $t_k := k \cdot \Delta$, for $k = 0, \dots, N$. Let $A_k = A(t_k)$, $b_k = b(t_k)$ and $x_k = x(t_k)$.

The resulting discretized system becomes:

$$x_{k+1} - x_k = \Delta(A_k x_k + b_k), \;\; k = 1, \dots, N.$$

This can be reformulated as an $\mathcal{A}\tilde{x} = \tilde{b}$ linear system of equation, where $\mathcal{A}$ is a block matrix

$$\mathcal{A} = \begin{pmatrix} I & & & & \\ -(I + \Delta A_1) & I & & & \\ & -(I + \Delta A_2) & I & & \\ & & & \ddots & \\ & & & -(I + \Delta A_{N-1}) & I \end{pmatrix},$$

$$\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \dots, \tilde{x}_N)^T,$$

and

$$\tilde{b} = \begin{pmatrix} (I + \Delta A_0)x_0 + \Delta \cdot b_0 \\ \Delta \cdot b_1 \\ \Delta \cdot b_2 \\ \\ \vdots \\ \\ \Delta \cdot b_{N-1} \end{pmatrix}.$$

We state without proof that in the special case when $N = 1$, $A(t) = a \in \mathbb{C}$ is constant and $Re(a) < 0$, then the condition number of $\mathcal{A}$ in this case is $\kappa(\mathcal{A}) = O(1/\Delta)$. Hence, the running time in this special case is $O\big((1/\Delta) \cdot \log N\big)$.

3 Quantum linear regression

In this part, we refer to the problem defined in Section 1.6. The only difference is that we examine the case where the data matrix X of size $N \times d$ is complex valued. Note, that Def. 5 (on page 19) of pseudo-inverse covered this case as well.

The other notations remain the same: d denotes the number of adjustable parameters (e.g. the dimension of the linear space in which we search for the best model), N is the size of the data set and $y \in \mathbb{R}^d$ is the vector containing the outputs. As stated in Section 1.4, our goal is to output the optimal β denoted by $\hat{\beta}$, but now in a quantum state: $\left|\hat{\beta}\right\rangle = X^+ |y\rangle = (X^*X)^{-1}X^* |y\rangle$.

The classical linear regression approach left us with an algorithm which runs in $poly(N, d)$. In this chapter, I present two algorithms which run in $poly(\log(N), d)$. The condition number, κ, of the data matrix and the desired precision ε also turn up in the running time, but the dependency is polynomial.

3.1 The initial approach

In this section, I give a high-level overview of the chronologically first quantum algorithm ([17]) that outputs $\left|\hat{\beta}\right\rangle$. This one will be quite similar to HHL-algorithm (demonstrated in Section 2.1): the only subroutine we use is the general template of HHL (see Figure 2). For this reason, I will delve deeper into a more efficient procedure in Section 3.2. Within this whole chapter, I follow the traces of article [9], [16] and [17].

Let' us start with the assumptions. Using the same trick as in Eq. (16), one can assume that X is a square ($N \times N$) Hermitian matrix. Similarly, we also live the assumption that y can be loaded to a quantum state $|y\rangle = \sum_{j=1}^{N} y_j |j\rangle$ efficiently. Denoting the eigenvectors and the respective eigenvalues of X by $|u_j\rangle$ and λ_j, we can observe that:

$$X^* = \sum_{j=1}^{N} \lambda_j |u_j\rangle \langle u_j| = X, \tag{27}$$

and

$$X^*X = \sum_{j=1}^{N} \lambda_j^2 |u_j\rangle \langle u_j| . \tag{28}$$

Here, we just used the singular-value decomposition of matrix X and the fact that X is Hermitian.

We begin the algorithm with the state $|y\rangle$. The procedure, as a flow diagram, will look like this:

$$|y\rangle \to X^* |y\rangle \to (X^*X)^{-1}X^* |y\rangle = \left|\hat{\beta}\right\rangle .$$

To realize this process, we have to call the general template of HHL (Figure 2) as a subroutine twice.

First, we would like to implement X^*. Note that we are not capable of doing this directly, as it is not necessarily a unitary matrix. For this purpose, we apply the process demonstrated in Figure 2. The step eigenvalue transformation (EVT) has to be set, so that the amplitude of $|1\rangle$ should be λ_j and the amplitude of $|0\rangle$ will be $\sqrt{1-\lambda_j^2}$, consecutively. After this, we get the state $X^* |y\rangle$.

Second, we apply $(X^*X)^{-1}$, which requires an analogue trick, but the amplitude of $|1\rangle$ will be λ_j^2. This leaves us with the desired state $\left|\hat{\beta}\right\rangle$.

The running time of this method is also polynomial in $\log(N)$.

3.2 A newer approach

Before delving deep into the algorithm, we define some frequently-used features of matrices and vectors which can be decisive for the speed. These will only pop up later, at the running time analysis.

Definition 9. *Let* $\rho(y) = \dfrac{\max_{1\leq j\leq N} |y_j|}{\sqrt{\frac{1}{N}\sum\limits_{j=1}^{N} |y_j|^2}}$, *where* $y \in \mathbb{C}^N$. *We say that* y *is balanced, if* $\rho(y) = O(1)$.

For example, if $y_1 = \ldots = y_N$, then $\rho(y) = 1$, so this is balanced according to the definition. However, in case $y_1 \neq 0$ and $y_2 = \ldots = y_N = 0$, then $\rho(y) = \sqrt{N}$, which is not balanced.

A similar parameter for matrices is the following.

Definition 10. *Let* $\sigma(A) = \dfrac{\max_{1\leq j\leq N} \|a_j\|}{\sqrt{\frac{1}{N}\sum\limits_{j=1}^{N} \|a_j\|^2}}$, *where* $A \in \mathbb{C}^{N\times N}$ *and* a_j *is the* j*-th row of* A *and* $\|.\|$ *denotes the conventional* ℓ_2*-norm. We say that* A *is balanced, if* $\sigma(A) = O(1)$.

For example, if $A = I_N$, then $\sigma(A) = 1$, which means that A is balanced.

However, if the first row of A is the all-1 vector, the other rows are the all-0 vectors, then $\sigma(A) = \sqrt{N}$. In this case, matrix A is not balanced.

Definition 11. *Let* $\tau(A, y) = \dfrac{\|\pi(A)y\|^2}{\|y\|^2}$, *where* $\pi(A)$ *is the projection onto the column space of* A.

Note that $\tau(A, y)$ is always less than or equal to 1 for any A, y.

Remark. *Note that* $\rho(y), \sigma(A) \geq 1$, *for any* A *and* y, *as the numerator is always greater than or equal to the denominator in both definitions.*

On the contrary with the previous part, here the data matrix X, whose pseudo-inverse we want to calculate, has size $N \times d$.

The main algorithm uses three subroutines:

- **Subroutine 1** approximates $|\hat{\beta}_i|$ for any $i \in \{1, \dots, d\}$,
- **Subroutine 2** approximates $|\hat{\beta}_i - \hat{\beta}_i|$ for any $i, j \in \{1, \dots, d\}$,
- **Subroutine 3** determines if β or $-\beta$ is δ-close to $\hat{\beta}$, if it is certain that one of the two cases holds.

In order to implement the first subroutine, we will need a lemma.

Lemma 3.1. *([7]) Let $\varepsilon > 0$ and define the following variables: $\delta_y = \Theta\left(\varepsilon/\sqrt{\log\left(\frac{\kappa}{\epsilon}\right)}\right)$, $\delta_z = \Theta\left(1/\kappa\sqrt{\log\left(\frac{\kappa}{\epsilon}\right)}\right)$, $J := \Theta\left(\frac{\kappa}{\varepsilon}\log\left(\frac{\kappa}{\varepsilon}\right)\right)$, $K := \Theta\left(\kappa\log\left(\frac{\kappa}{\varepsilon}\right)\right)$. Let the function $h(x)$ be defined as:*

$$h(x) = \sum_{j=0}^{J}\sum_{k=-K}^{K} \frac{i}{\sqrt{2\pi}} k\delta_y\delta_z^2 \cdot e^{k^2\delta_z^2/2} \cdot e^{-i\cdot x\cdot jk\delta_y\delta_z}.$$

Then, $|h(x) - 1/x| \leq \varepsilon$ for $x \in [-1, -1/\kappa] \cup [1/\kappa, 1]$.

The core idea behind the lemma is to rewrite the function $1/x$ as a weighted double integral and approximate the integrals by finite sums. The precise calculations can be found in article ([14]).

The idea can be extended to matrices as the following corollary claims.

Corollary 3.1.1. *Let $\varepsilon > 0$ and A a matrix whose nonzero eigenvalues are in the range $D_\kappa := [-1, -1/\kappa] \cup [1/\kappa, 1]$. Then $\|h(A) - A^+\| \leq \varepsilon$.*

Now, let's move on to the first subroutine.

Subroutine 1 for ε-approximating $|\hat{\beta}_i|$ for any $i \in \{1, \dots, d\}$.

The process consists of four steps. First, we apply a trick which already turned up in this thesis: instead of the matrix X, we will use a broader block matrix by which trick we can handle the nasty cases. Next, we give a function which helps us approximate the desired outcome. Third, we introduce operators that will be applied to our input $\left|0^{m+2}\right\rangle|y\rangle$. Last, we measure the obtained state which gives us a probability that contains the sought-after value.

Let the singular value decomposition of X be:

$$X = \sum_{j=1}^{d} s_j |u_j\rangle\langle v_j|,$$

where $s_j \in [1/\kappa, 1]$, $u_j \in \mathbb{R}^{\mathsf{N}}$, $v_j \in \mathbb{R}^{\mathsf{d}}$ are of unit length, for all $j \in \{1, 2, \dots, d\}$.

In order to eliminate the cases where X is a non-square matrix or not Hermitian, we introduce the following matrix:

$$A = \begin{pmatrix} 0 & X \\ X^T & 0 \end{pmatrix},$$

and

$$|b\rangle := |0\rangle\,|y\rangle\,.$$

Then, the spectral decomposition of A looks as follows: $A = \sum\limits_{j=1}^{d} s_j\,|+_j\rangle\,\langle +_j| + \sum\limits_{j=1}^{d} s_j\,|-_j\rangle\,\langle -_j|$, where $|\pm\rangle = \frac{1}{\sqrt{2}}\big(\,|0\rangle\,|u_j\rangle \pm |1\rangle\,|v_j\rangle\,\big)$. Let $D_k = [-1, -1/\kappa] \cup [1/\kappa, 1]$ denote the range in which the nonzero eigenvalues of A lie in.
A simple substitution shows that $A^{+}\,|b\rangle = |1\rangle\,\left|\hat{\beta}\right\rangle$, which implies that

$$\langle 1| \otimes \langle i| \cdot A^{+}\,|b\rangle = \hat{\beta}_i. \tag{29}$$

In order to implement operation A^{+}, we will use the function h defined in Lemma (3.1) and keep the same notations. From Cor. (3.1.1), we know that $\|h(A) - A^{+}\| \leq \varepsilon$.

Thus,

$$\|h(A)\,|b\rangle - A^{+}\,|b\rangle\,\| \leq \varepsilon, \tag{30}$$

using the fact the $|b\rangle$ is a unit vector.

For simplicity, we introduce two new variables such that

$$|z\rangle := A^{+}\,|b\rangle = |1\rangle\,\left|\hat{\beta}\right\rangle,$$

and

$$\left|z'\right\rangle := h(A)\,|b\rangle\,.$$

Then, the following short calculation shows that the i^{th} coordinate of the second chunk of $|z'\rangle$ is ε-close to $\left|\hat{\beta}_i\right\rangle$, for any $i \in \{1, 2, \ldots, d\}$.

$$|\,\langle 1| \otimes \langle i| \cdot \left|z'\right\rangle - \hat{\beta}_i| = |\,\langle 1| \otimes \langle i| \cdot \left|z'\right\rangle - |\,\langle 1| \otimes \langle i| \cdot |z\rangle\,| \leq \|\,\left\langle z'\right| - \langle z|\,\| = O(\varepsilon), \tag{31}$$

The last equation comes from (30).

In order to get the i-th component of the least-squares fit, we need to ε-approximate $\big(\langle 1| \otimes \langle i|\big) \cdot |z\rangle$, for which it is sufficient to give an ε-approximation of $\big(\,|1\rangle \otimes |i\rangle\,\big) \cdot \langle z'|$.

To simplify the formulas, we use the following notations: $\alpha(j,k) := \frac{i}{2\pi}k\delta_y\delta_z^2 e^{-k^2\delta_z^2/2}$, $\alpha := \sum\limits_{j=0}^{J-1}\sum\limits_{k=-K}^{K} |\alpha(j,k)|$ and $\nu(j,k) = jk\delta_z\delta_y$.

For approximating the β_i, we introduce new operators. Let $U := W^{\star}VW$, where

- $W : |0^m\rangle \to \frac{1}{\sqrt{\alpha}} \sum\limits_{j=0}^{J-1}\sum\limits_{k=-K}^{K} \sqrt{|\alpha(j,k)|}\,|j,k\rangle$, $m \in O(log(\kappa/\varepsilon)))$,
- $V = i\sum\limits_{j=0}^{J-1}\sum\limits_{k=-K}^{K} |j,k\rangle\,\langle j,k| \otimes \operatorname{sign}(k)e^{-iA\nu(j,k)}$.

In the description of this procedure, this is the only time when i denotes the imaginary unit, not the index.

It follows from the definitions that

$$U|0^m\rangle|b\rangle = \frac{1}{\alpha}|0^m\rangle h(A)|b\rangle + |\lambda\rangle = \left(\frac{\|h(A)|b\rangle\|}{\alpha}\right)|0^m\rangle\frac{h(A)|b\rangle}{\|h(A)|b\rangle\|} + |\lambda\rangle, \tag{32}$$

where $|\lambda\rangle$ satisfies the equation of $\big(|0^m\rangle\langle 0^m|\otimes I\big)|\lambda\rangle = 0$.

We need another unitary operator R such that:

- $R : |0\rangle|1,i\rangle \to |1\rangle|1,i\rangle,$
- $R : |0\rangle|1,j\rangle \to |0\rangle|1,j\rangle, \quad 1 \le j \le N, \;\; j \ne i,$
- $R : |0\rangle|0,j\rangle \to |0\rangle|0,j\rangle, \quad 1 \le j \le N.$

The way operator R works is pretty similar to what controlled operators do. Note that R is specific to the fixed index i, so it shouldn't necessarily be the same for different indices.

The following equation sums up what R does:

$$R\left(\sum_{k=1}^{N}|0\rangle\big(|1,k\rangle + |0,k\rangle\big)\right) = |1\rangle|1,i\rangle + \sum_{k\ne i}|0\rangle|1,k\rangle + \sum_{k=1}^{N}|0\rangle|0,k\rangle. \tag{33}$$

By equations (32) and (33), we get that:

$$R_{2,3}U_{1,3}|0^m\rangle_1|0\rangle_2|b\rangle_3 =$$
$$= \frac{\langle 1,i|z'\rangle}{\alpha}|0^m\rangle_1|1\rangle_2|1,i\rangle_3 + \sum_{i'\ne i}\frac{\langle 1,i|z'\rangle}{\alpha}|0^m\rangle_1|0\rangle_2|1,i'\rangle_3 + \sum_{j}\frac{\langle 0,j|z'\rangle}{\alpha}|0^m\rangle_1|0\rangle_2|0,j\rangle_3.$$

After applying these operators, we make a measurement on the first $m+1$ qubits. The probability of getting $|0^m1\rangle$ as an output is obviously

$$p := \frac{1}{\alpha^2}\cdot|\big\langle 1,i\big|z'\big\rangle|^2, \tag{34}$$

as this is the square of the respective amplitude of the first two register.

Using Amplitude Estimation, we can $\Theta\left(\frac{\varepsilon^2}{\kappa^2}\right)$-approximate p by $\tilde{p}$. This will output $\alpha\sqrt{\tilde{p}}$, which is an $O(\varepsilon)$-approximation of $|\langle 1,i|z'\rangle|$, since $\Big||\langle 1,i|z'\rangle| - \alpha\sqrt{\tilde{p}}\Big| = \Big|\alpha\sqrt{p} - \alpha\sqrt{\tilde{p}}\Big| = O(\varepsilon)$.

Complexity:

The above algorithm has a probability bound 3/4. In order to reach $1-\delta$, we repeat the same process $O(\log(1/\delta))$ times.

The exact way of implementing V and W can be found in ([5]), here we just make use of those results.

The implementation V needs $O\big(d^{3/2}\cdot poly(\log(\kappa/\varepsilon))\big)$ uses of P_X and W needs $O\big(\kappa\cdot poly(\log(\kappa/\varepsilon))\big)$. For R, only $poly(\log(N))$ time is needed, trivially. To prepare $|y\rangle$, $O(\log(\kappa/\varepsilon))$ uses of P_y is enough.

Therefore, the number of uses of P_X and P_y for this subroutine is:

$$O\bigg(\frac{d^{3/2}\kappa^3}{\varepsilon^2}\cdot poly\bigg(\log\Big(\frac{\kappa}{\varepsilon}\Big)\log\Big(\frac{1}{\delta}\Big)\bigg)\bigg).$$

The $\left(\dfrac{\kappa}{\epsilon}\right)^2$ factor comes from the Amplitude Estimation algorithm.

We assumed in Chapter 1.1 that the implementation of P_X and P_y can be constructed in polynomial time of the logarithm of their size, therefore the total complexity of this subroutine is

$$O\bigg(\frac{d^{3/2}\kappa^3}{\varepsilon^2}\cdot poly(\log(N))\cdot poly\bigg(\log\Big(\frac{\kappa}{\varepsilon}\Big)\log\Big(\frac{1}{\delta}\Big)\bigg)\bigg).$$

Subroutine 2 for ε-approximating $|\hat{\beta}_i-\hat{\beta}_j|$ for any $i,j\in\{1,\dots,d\}$.

Here, we follow a similar route as in Subroutine 1: we introduce a new operator Q, apply $Q\cdot W^*VW$ to the same initial state, make a measurement and apply Amplitude Estimation.

Let us introduce the notation $|1,\pm_{i,j}\rangle := \frac{1}{\sqrt{2}}\cdot|1\rangle\otimes(|i\rangle\pm|j\rangle)$ and define unitary operator Q which is a controlled-gate and satisfies the four properties below:

- $Q\,|0\rangle\,|1,-_{i,j}\rangle = |1\rangle\,|1,-_{i,j}\rangle$
- $Q\,|0\rangle\,|1,+_{i,j}\rangle = |1\rangle\,|1,+_{i,j}\rangle$
- $Q\,|0\rangle\,|1,l\rangle = |0\rangle\,|1,l\rangle$ where $1\le l\le N$ and $l\ne i,j$
- $Q\,|0\rangle\,|0,k\rangle = |0\rangle\,|0,k\rangle$ where $1\le k\le N$.

Note that Q is specific to two indices, i and j. If we choose another two indices when setting the problem, operator Q might not be the same in that case.

The algorithm consists of two steps: applying operators $(Q\cdot U)$ and measuring.

By equation (32) and the above properties of Q, we get that:

$$\begin{aligned} Q_{2,3}U_{1,3}\,|0^m\rangle_1\,|0\rangle_2\,|b\rangle_3 = {} & \frac{\langle 1,-_{i,j}|z'\rangle}{\alpha}\,|0^m\rangle_1\,|1\rangle_2\,|1,-_{i,j}\rangle_3 + \frac{\langle 1,+_{i,j}|z'\rangle}{\alpha}\,|0^m\rangle\,|0\rangle\,|1,+_{i,j}\rangle + \\ & \sum_{l\ne i,j}\frac{\langle 1,l|z'\rangle}{\alpha}\,|0^m\rangle_1\,|0\rangle_2\,|1,l\rangle_3 + \sum_k\frac{\langle 0,k|z'\rangle}{\alpha}\,|0^m\rangle_1\,|0\rangle_2\,|0,k\rangle_3\,. \end{aligned}$$

The probability of getting $|0^m\rangle\,|1\rangle$ after measuring the first $m+1$ qubits is

$$\frac{|\,\langle 1,-_{i,j}|z'\rangle\,|^2}{\alpha^2} =: p''. \tag{35}$$

The key is the following:

$$|\hat{\beta}_i-\hat{\beta}_j| = \sqrt{2}|\,\langle 1,-_{i,j}|z\rangle\,| \approx \sqrt{2}|\,\langle 1,-_{i,j}|z'\rangle\,| = \sqrt{2}\alpha\sqrt{p''}, \tag{36}$$

where in the last equation we used (35).

Using Amplitude Estimation we can get an $O(\varepsilon/\alpha)$-additive approximation of p''.

Complexity:

Operator Q is practically a $CNOT$-gate, therefore $O(poly(\log(N))$ time is enough to construct it. The implementation of V and W was covered at the end of Subroutine 1. Hence, this algorithm makes

$$O\Big(\frac{d^{3/2}\kappa^3}{\varepsilon^2}\cdot poly\Big(\log\Big(\frac{\kappa}{\varepsilon\delta}\Big)\Big)\Big)$$

uses of P_X and P_y.

Taking the complexity of constructing P_X and P_y into consideration, we get the complexity of Subroutine 2:

$$O\Big(poly(\log(N))\frac{d^{3/2}\kappa^3}{\varepsilon^2}\cdot poly\Big(\log\Big(\frac{\kappa}{\varepsilon\delta}\Big)\Big)\Big).$$

Subroutine 3 decides Suppose $\beta \in \mathbb{R}^d$ is give such that either $\|\beta+\hat{\beta}\| \le \delta$ or $\|\beta-\hat{\beta}\| \le \delta$ for some $\delta < \frac{\tau}{2\sigma\rho\sqrt{d}}$, where $\tau = \tau(X,y), \sigma = \sigma(X)$ and $\rho = \rho(y)$.

This part consists of two major pieces. First, we define a quantity which is a function of β. Estimating this with a proper mistake will tell us which of the two cases holds. Next, we give an algorithm to estimate this quantity. For this, we apply operator G (defined later), making a measurement and using Amplitude Estimation.

First, we will obtain several upper bound estimations that will make sense when distinguishing the two cases.

Let $\hat{y}$ denote $X\hat{\beta}$. Then, $\tau = \|\hat{y}\|^2$ holds true. Combining all the information, we obtain the following estimation.

$$\sqrt{\tau} \le \|\hat{\beta}\| \le \|X^+y\| = \|X^+\hat{y}\| \le \kappa\sqrt{\tau}. \tag{37}$$

In (11), we showed that

$$\|x_i\| \le \frac{\sigma\sqrt{d}}{\sqrt{N}}, \quad 1 \le i \le N. \tag{38}$$

Equations (37) and (38) imply that

$$|x_i^T\hat{\beta}| \le \frac{\sigma\kappa\sqrt{\tau d}}{\sqrt{N}}, \quad 1 \le i \le N. \tag{39}$$

Using the fact that $\|y\| = 1$ and $\rho(y) = \rho$, we obtain that

$$|y_i| \le \frac{\rho}{\sqrt{N}}, \quad 1 \le i \le N. \tag{40}$$

Let $\hat{q}_i = y_i \cdot x_i^T\hat{\beta}$ and $q_i = y_i \cdot x_i^T\beta$ for $i = 1,\dots,N$. A quick calculation shows that

$$\sum_{i=1}^N \hat{q}_i = \tau. \tag{41}$$

By (39) and (40), a trivial upper bound for $|\hat{q}_i|$ $\big(1 \le i \le N\big)$ can be stated:

$$|\hat{q}_i| \le \frac{\rho\sigma\kappa\sqrt{\tau d}}{N}. \tag{42}$$

Now, we go on to prove that the sum of q_i-s differs significantly in the two cases. For this purpose, we will use all of the calculations we have done above.

First case: $\|\beta - \hat{\beta}\| \le \delta < \dfrac{\tau}{2\rho\sigma\sqrt{d}}$.

Using the equations (38) and (40) we get

$$|q_i - \hat{q}_i| \le |y_i| \|x_i\| \|\beta - \hat{\beta}\| \le \frac{\rho}{\sqrt{N}} \frac{\sigma\sqrt{d}}{\sqrt{N}} \delta < \frac{\tau}{2N}, \quad 1 \le i \le N. \tag{43}$$

Combining equations (41) and (43), we obtain that:

$$\sum_{i=1}^{N} q_i \ge \sum_{i=1}^{N} \hat{q}_i - \sum_{i=1}^{N} |q_i - \hat{q}_i| > \tau - \frac{\tau}{2} = \frac{\tau}{2}. \tag{44}$$

Second case: $\|\beta + \hat{\beta}\| \le \delta < \dfrac{\tau}{2\rho\sigma\sqrt{d}}$.

Equations (38) and (40) imply that

$$|q_i + \hat{q}_i| \le |y_i| \|x_i\| \|\beta + \hat{\beta}\| \le \frac{\rho}{\sqrt{N}} \frac{\sigma\sqrt{d}}{\sqrt{N}} \delta < \frac{\tau}{2N}, \quad 1 \le i \le N. \tag{45}$$

Equations (41) and (45) give the following upper bound:

$$\sum_{i=1}^{N} q_i \le -\sum_{i=1}^{N} \hat{q}_i + \sum_{i=1}^{N} |q_i + \hat{q}_i| < -\tau + \frac{\tau}{2} = -\frac{\tau}{2}. \tag{46}$$

Knowing from equation (44) and (46) that $\sum\limits_{i=1}^{N} q_i$ is either strictly less than $-\dfrac{\tau}{2}$ or strictly more than $\dfrac{\tau}{2}$, all that remains is to estimate $\sum\limits_{i=1}^{N} q_i$ up to an additive error of $\tau/2$.

For this purpose, we introduce operator G which works as follows:

$$G : |i\rangle |0\rangle \to |i\rangle |\psi_i\rangle ,$$

where

$$|\psi_i\rangle := \sqrt{\frac{1}{2} + \frac{Nq_i}{4\sigma\rho\kappa\sqrt{d}}}\ |0\rangle + \sqrt{\frac{1}{2} - \frac{Nq_i}{4\sigma\rho\kappa\sqrt{d}}}\ |1\rangle .$$

In order to prove that G is a valid operator, we must make sure that

$$|q_i| < \frac{2\sigma\rho\kappa\sqrt{d}}{N}. \tag{47}$$

In the first case, we meet the requirement of (47):

$$|q_i| < |\hat{q}_i| + |q_i - \hat{q}_i| \leq \frac{\sigma\rho\kappa\sqrt{\tau d}}{N} + \frac{\tau}{2N} \leq \frac{2\sigma\rho\kappa\sqrt{d}}{2N} + \frac{1}{2N} \leq \frac{2\sigma\rho\kappa\sqrt{d}}{N}.$$

Here we used equations (42) and (43), then the fact that $\tau \leq 1$ and $\rho, \sigma, \kappa, d \geq 1$.

For the second case, we use equation (42) and (45), but it works in a pretty similar way:

$$|q_i| < |\hat{q}_i| + |q_i + \hat{q}_i| \leq \frac{2\sigma\rho\kappa\sqrt{d}}{N}.$$

In order to get a $\tau/2$-estimation of $\sum_{i=1}^{N} q_i$, we apply G to $\left(\frac{1}{\sqrt{N}}\sum_{i=1}^{N} |i\rangle\right)|0\rangle$, which produces the state $\frac{1}{\sqrt{N}}\sum_{i=1}^{N} |i\rangle|\psi_i\rangle$. The probability of obtaining 0 after measuring the second register is

$$p := \frac{1}{2} + \frac{\sum_{i=1}^{N} q_i}{4\sigma\rho\kappa\sqrt{d}}\ .$$

Using Amplitude Estimation, we can get an $O(\kappa^{-1}d^{-1/2})$ approximation of p in the form of a quantum state.

Complexity:

To implement G, the most straightforward way is the following:

from $|i\rangle|0\rangle$, where $i \in [N]$, the state $|i\rangle|0\rangle|q_i\rangle$ can be prepared by $O(d)$ uses of P_X and P_y. Having q_i in the last register, we can get the state $|i\rangle|\psi_i\rangle|q_i\rangle$. Last, we diminish q_i from the last register which takes $O(d)$ uses of P_X and P_y.

Considering that the construction of P_X and P_y takes $O(poly(\log(N))$ runtime, the total complexity of this subroutine is

$$O\big(d^{3/2}\kappa \cdot poly(\log N)\big).$$

Having demonstrated the subroutines we need, let's move on to describing the algorithm itself.

Algorithm

Step 1: Subroutine 1 for all $j \in \{1, \dots, d\}$.

Let μ_j denote the estimation of β_j we got. The remaining steps are about setting the signs of these. Let Υ be the set of indices defined as: $\Upsilon := \big\{j \in \{1, \dots, d\} : \mu_j > \frac{2}{3}\varepsilon'\big\}$.

Step 2: Pick an arbitrary $j_0 \in \Upsilon$ and run Subroutine 2 with j_0 for all $j \in \Upsilon$.

Let γ_j denote the estimation of $|\hat{\beta}_{j_0} - \hat{\beta}_j|$.

Step 3: Define the signs by the following logic:

- if $j \notin \Upsilon$, then $s_j = 0$,
- if $j = j_0$, then $s_j = 1$,
- if $j \in \Upsilon \setminus j_0$ and $\big||\mu_{j_0} - \mu_j| - \gamma_j\big| \leq \varepsilon'/2$, then $s_j = 1$,
- if $\Upsilon \setminus j_0$ and $\big||\mu_{j_0} - \mu_j| - \gamma_j\big| > \varepsilon'/2$, then $s_j = -1$.

Let $\beta' \in \mathbb{R}^d$, where $\beta'_j := s_j \mu_j$ for all $j = 1, \dots, d$.

Step 4: Apply Subroutine 3 with β'.

Having performed these four steps, the algorithm outputs either $\beta' \in \mathbb{R}^d$ or -$\beta' \in \mathbb{R}^d$.

Now, let's move on to determine the runtime of the algorithm described above.

Complexity

Recall, that $\varepsilon' = \min\left(\frac{\tau}{2\sigma\rho d}, \varepsilon\right)$ and $\delta = \min(1/d, \varepsilon)$, where $\tau \in \Omega(1)$, $\sigma, \rho \in O(1)$, implying that $\varepsilon' \in \Omega(\delta)$.

Let's calculate the complexity of the whole algorithm.

Subroutine 1 is called $O(d)$ times (Step 1), therefore the complexity of this part is

$$O\left(\frac{d^{2.5}\kappa^3}{\delta^2} \cdot poly(\log(N)) \cdot poly\left(\log\left(\frac{\kappa d}{\delta}\right)\right)\right). \tag{48}$$

Subroutine 2 is used $O(d)$ times (Step 2), resulting the complexity of

$$O\left(\frac{d^{2.5}\kappa^3}{\delta^2} \cdot poly(\log(N)) \cdot poly\left(\log\left(\frac{\kappa d}{\delta}\right)\right)\right). \tag{49}$$

Subroutine 3 is invoked constant times (Step 4), resulting

$$O\Bigg(\kappa d^{1.5} \cdot poly(\log(N))\Bigg) \tag{50}$$

runtime.

Combining (48), (49) and (50) the complexity of the whole algorithm is:

$$O\Bigg(poly(\log(N)) \cdot \frac{d^{2.5}\kappa^3}{\delta^2} \cdot poly\Bigg(\log\Bigg(\frac{\kappa d}{\delta}\Bigg)\Bigg)\Bigg).$$

Theorem 3.2. *The above algorithm gives a correct estimation of $\hat{\beta}$.*

Proof. The proof can be partitioned into four major chunks. First, the some estimations are listed, which are used in the latter parts. Next, we prove that S is not empty. Then, we check that either all coordinates of β' has the same sign as the respective coordinate of $\hat{\beta}$ or they all have an opposite sign. Last, we make sure that the vector β' or $-\beta'$ is ε-close to the sought-after β.

To begin with, let me collect some of the most important estimations that will be utilized in the proof. We will only care about cases where the three subroutines succeed. The probability of this happening is at least 2/3.

First of all, the following two equations hold as we did so:

$$\Big|\mu_i - |\hat{\beta}_i|\Big| \leq \frac{\varepsilon'}{6}, \qquad \forall i = 1, \dots, d, \tag{51}$$

and

$$\Big|\gamma_j - |\hat{\beta}_{j_0} - \hat{\beta}_j|\Big| \leq \frac{\varepsilon'}{6}, \qquad j \in \Upsilon,\ j \neq j_0. \tag{52}$$

It is straightforward to verify that:

$$-\Big|\mu_i - |\hat{\beta}_i|\Big| \leq |\hat{\beta}_i| - |\mu_i| \leq \Big|\mu_i - |\hat{\beta}_i|\Big|. \tag{53}$$

By (53) and the definition of Υ, we obtain the next two bounds:

$$|\hat{\beta}_j| \geq |\mu_j| - \Big|\mu_j - |\hat{\beta}_j|\Big| \geq \frac{2\varepsilon'}{3} - \frac{\varepsilon'}{6} = \frac{\varepsilon}{2}, \quad j \in \Upsilon, \tag{54}$$

and

$$|\hat{\beta}_j| \leq |\mu_j| + \Big|\mu_j - |\hat{\beta}_j|\Big| \leq \frac{2\varepsilon'}{3} + \frac{\varepsilon'}{6} = \frac{5\varepsilon}{6}, \quad j \notin \Upsilon. \tag{55}$$

In the next steps, we will prove that the set Υ is not empty. In equation (37), we showed that $\|\hat{\beta}\| \geq \sqrt{\tau}$. This implies that there exists an $i_0 \in \{1, \dots, d\}$ such that

$$|\hat{\beta}_{i_0}| \geq \sqrt{\frac{\tau}{d}} \geq \frac{\tau}{\sigma \rho d} \geq 2\varepsilon'. \tag{56}$$

In the first estimation we used that $\|\hat{\beta}\| = \sqrt{\sum\limits_{i=1}^{d} |\hat{\beta}_i|^2}$. The second one is true due to the fact that $\rho, \sigma, d \geq 1$ and $\tau \leq 1$. Last, the definition of ε' was exploited.

Equations (53) and (56) yield

$$\mu_{i_0} \;\geq\; |\hat{\beta}_{i_0}| - \Big|\mu_{i_0} - |\hat{\beta}_{i_0}|\Big| \;\geq\; 2\varepsilon' - \frac{\varepsilon'}{6} \;>\; \frac{2\varepsilon'}{3}.$$

This means that i_0 is an element of Υ.

Now, we prove that

$$s_j = sgn(\hat{\beta}_j) \cdot sgn(\beta_j), \quad \forall j \in \Upsilon. \tag{57}$$

To remind, the left side of this equation was defined as $s_j = sgn(\beta'_j)$. This trivially implies that either

$$sgn(\beta'_j) = sgn(\hat{\beta}_j), \quad \forall j \in \Upsilon,$$

or

$$sgn(\beta'_j) = -sgn(\hat{\beta}_j), \quad \forall j \in \Upsilon$$

holds.

To verify (57), we have to inspect three cases.

1. If $j = j_0$, then (57) trivially holds.

2. If $j \neq j_0$ and $sgn(\hat{\beta}_j) = sgn(\beta_{j_0})$, we have the following sequence of estimations:

$$\Big||\mu_{j_0} - \mu_j| - \gamma_j\Big| \leq \Big|\mu_{j_0} - |\hat{\beta}_{j0}|\Big| + \Big|\mu_j - |\hat{\beta}_j|\Big| + \Big|\gamma_j - |\hat{\beta}_{j_0} - \hat{\beta}_j|\Big| \leq 3 \cdot \frac{\varepsilon'}{6} = \frac{\varepsilon'}{2}.$$

According to the algorithm, the sign, s_j was set to 1, which justifies (57).

3. The last case, where $j \neq j_0$ and $sgn(\hat{\beta}_j) = -sgn(\beta_{j_0})$ is somewhat similar.

$$\Big|\hat{\beta}_{j_0} - \hat{\beta}_j\Big| = \Big||\hat{\beta}_{j_0}| - |\hat{\beta}_j|\Big| + 2 \cdot \min\{|\hat{\beta}_{j_0}|, |\hat{\beta}_j|\} > \Big||\hat{\beta}_{j_0}| - |\hat{\beta}_j|\Big| + \varepsilon'. \tag{58}$$

In the last step, we exploited (53) for j and j_0.

From (51), (52) and (58), we have the following:

$$\Big||\mu_{j_0} - \mu_j| - \gamma_j\Big| \geq \varepsilon' - \left(\Big|\mu_{j_0} - |\hat{\beta}_{j_0}|\Big| + \Big|\mu_j - |\hat{\beta}_j|\Big| + \Big|\gamma_j - |\hat{\beta}_{j_0} - \hat{\beta}_j|\Big|\right) > \varepsilon' - 3 \cdot \frac{\varepsilon'}{6} = \frac{\varepsilon'}{2}.$$

This implicates $s_j = -1$, which confirms (57).

What remains is to prove that either β' or $-\beta'$ is ε-close to $\hat{\beta}$.

It is enough to see that either

$$\|\beta' - \hat{\beta}\|_\infty < \varepsilon', \tag{59}$$

or

$$\|\beta' + \hat{\beta}\|_\infty < \varepsilon', \tag{60}$$

as it results that either

$$\|\beta' - \hat{\beta}\| < \sqrt{d}\varepsilon' \leq \frac{\tau}{2\sigma\rho\sqrt{d}},$$

or

$$\|\beta' + \hat{\beta}\| < \sqrt{d}\varepsilon' \leq \frac{\tau}{2\sigma\rho\sqrt{d}},$$

according to the definition of ε'.

For this purpose, we observe two cases.

First, if $j \notin \Upsilon$, then

$$|\beta'_j - \hat{\beta}_j| = |\beta'_j + \hat{\beta}_j| \leq \frac{5\varepsilon'}{6}.$$

Here, (55) was used and the fact that $s_j = 0$ for $j \notin \Upsilon$.

Second, if $j \in \Upsilon$, then $s_j \in \{\pm 1\}$, therefore $|\beta'_j| = |\mu_j|$. From (51), we know that

$$\Big| |\beta'_j| - |\hat{\beta}_j| \Big| \leq \frac{\varepsilon'}{6}.$$

If $sgn(\beta'_j) = sgn(\hat{\beta}_j)$, then

$$\Big| \beta'_j - \hat{\beta}_j \Big| \leq \frac{\varepsilon'}{6}. \tag{61}$$

If $sgn(\beta'_j) = -sgn(\hat{\beta}_j)$, then

$$\Big| \beta'_j + \hat{\beta}_j \Big| \leq \frac{\varepsilon'}{6}. \tag{62}$$

Since one of (61) and (62) is true, either (59) or (60) holds.

As $\varepsilon' \leq \varepsilon$, the output vector is close enough to the optimal $\hat{\beta}$.

□

References

[1] Aaronson, Scott. *Read the fine print.* Nature Physics 11.4 (2015): 291-293.

[2] Berry, Dominic W., et al. *Efficient quantum algorithms for simulating sparse Hamiltonians.* Communications in Mathematical Physics 270 (2007): 359-371.

[3] Berry, Dominic W. *High-order quantum algorithm for solving linear differential equations.* Journal of Physics A: Mathematical and Theoretical 47.10 (2014): 105301.

[4] Brassard, Gilles, et al. *Quantum amplitude amplification and estimation.* Contemporary Mathematics 305 (2002): 53-74.

[5] Childs, Andrew M., Robin Kothari, and Rolando D. Somma. *Quantum algorithm for systems of linear equations with exponentially improved dependence on precision.* SIAM Journal on Computing 46.6 (2017): 1920-1950.

[6] Clader, B. David, Bryan C. Jacobs, and Chad R. Sprouse. *Preconditioned quantum linear system algorithm.* Physical review letters 110.25 (2013): 250504.

[7] Coppersmith, Don. *An approximate Fourier transform useful in quantum factoring.* arXiv preprint quant-ph/0201067 (2002).

[8] Gharibian S.: *Lecture 4: Linear systems of equations and a BQP-complete problem*, Quantum Complexity Theory (lecture notes), University of Paderborn, Summer 2019.

[9] Golub, G. H., and C. F. Van Loan. *Matrix computations 4th edition the johns hopkins university press.* Baltimore, MD (2013).

[10] Harrow, Aram W., Avinatan Hassidim, and Seth Lloyd. *Quantum algorithm for linear systems of equations.* Physical review letters 103.15 (2009): 150502.

[11] Lin, Lin. *Lecture notes on quantum algorithms for scientific computation.* arXiv preprint arXiv:2201.08309 (2022).

[12] Low, Guang Hao, and Isaac L. Chuang. *Hamiltonian simulation by qubitization.* Quantum 3 (2019): 163.

[13] Mitarai, Kosuke, Masahiro Kitagawa, and Keisuke Fujii. *Quantum analog-digital conversion.* Physical Review A 99.1 (2019): 012301.

[14] Nielsen, Michael A., and Isaac Chuang. *Quantum computation and quantum information.* (2002): 558-559.

[15] Shende, Vivek V., Stephen S. Bullock, and Igor L. Markov. *Synthesis of quantum logic circuits.* Proceedings of the 2005 Asia and South Pacific Design Automation Conference. 2005.

[16] Wang, Guoming. *Quantum algorithm for linear regression.* Physical review A 96.1 (2017): 012335.

[17] Wiebe, Nathan, Daniel Braun, and Seth Lloyd. *Quantum algorithm for data fitting.* Physical review letters 109.5 (2012): 050505.

[18] De Wolf, R. *Quantum computing: Lecture notes.* arXiv preprint arXiv:1907.09415 (2019).